Meta Data

CIFAR-10 adalah dataset yang sering digunakan dalam bidang visi komputer dan pembelajaran mesin. Berikut adalah gambaran umumnya:

1. **Deskripsi**: CIFAR-10 merupakan singkatan dari Canadian Institute for Advanced Research (CIFAR) 10. Ini adalah subset berlabel dari dataset 80 juta gambar kecil. Dataset ini terdiri dari 60.000 gambar warna berukuran 32x32 piksel dalam 10 kelas, dengan 6.000 gambar per kelas.

2. **Kelas**: Dataset ini berisi gambar yang termasuk dalam 10 kelas berikut:

- Airplane
- Automobile
- Bird
- Cat
- Deer
- Dog
- Frog
- Horse
- Ship
- Truck

3. **Penggunaan**: CIFAR-10 sering digunakan untuk menguji algoritma dalam bidang pembelajaran mesin dan visi komputer, terutama untuk tugas-tugas seperti klasifikasi gambar, deteksi objek, dan segmentasi gambar. Ini sering digunakan sebagai dataset yang cocok bagi pemula untuk latihan dan pembelajaran teknik pembelajaran mesin karena ukurannya yang kecil dan gambar-gambar yang relatif sederhana.

4. **Format**: Setiap gambar dalam CIFAR-10 adalah gambar RGB 32x32 piksel, yang berarti memiliki tiga saluran warna (merah, hijau, dan biru). Dataset ini dibagi menjadi set pelatihan dengan 50.000 gambar dan set pengujian dengan 10.000 gambar.

5. **Tantangan**: Meskipun sederhana, CIFAR-10 menimbulkan beberapa tantangan bagi algoritma pembelajaran mesin karena ukuran gambar yang kecil, resolusi rendah, dan kemiripan visual antara kelas seperti kucing dan anjing, atau truk dan mobil.

Peneliti dan praktisi sering menggunakan CIFAR-10 untuk mengevaluasi kinerja berbagai model dan algoritma pembelajaran mesin, menjadikannya dataset benchmark standar dalam bidang visi komputer.

- Source: CIFAR homepage.

Import Library

In [1]:
```
# Installing Visual Keras

!pip install visualkeras
```

```
Collecting visualkeras
  Downloading visualkeras-0.0.2-py3-none-any.whl (12 kB)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (9.4.
0)
Requirement already satisfied: numpy>=1.18.1 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (1.25
.2)
Collecting aggdraw>=1.3.11 (from visualkeras)
  Downloading aggdraw-1.3.18-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (993 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 993.7/993.7 kB 9.0 MB/s eta 0:00:00
Installing collected packages: aggdraw, visualkeras
Successfully installed aggdraw-1.3.18 visualkeras-0.0.2
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')

import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout, BatchNormalization
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

from sklearn.model_selection import train_test_split

from keras.utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
```

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

# Visualizing our model (Hidden Input)
import visualkeras
```

Load CIFAR-10 dataset

Tensorflow sudah menyediakan dataset menjadi set pelatihan dengan 50.000 gambar dan set pengujian dengan 10.000 gambar.

X_train disini berarti adalah data image nya dan y_train adalah label dari image nya

In [3]:
```
# Load CIFAR-10 dataset

(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
```

Split Train set to Validation Set

In [4]:
```
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.1, random_state=70)
```

Let's check and make sure the data

- Training images shape: (45000, 32, 32, 3)
  - Ini berarti terdapat 45,000 gambar pelatihan.
  - Setiap gambar memiliki dimensi 32x32 piksel.
  - Angka 3 menunjukkan bahwa gambar memiliki tiga saluran warna (RGB: merah, hijau, biru).

In [5]:
```
# Explore the data
print("Training images shape:", X_train.shape)
print("Training labels shape:", y_train.shape)
print("\nTest images shape:", X_test.shape)
print("Test labels shape:", y_test.shape)
print("\nValidation images shape:", X_valid.shape)
print("Validation labels shape:", y_valid.shape)
```

```
Training images shape: (45000, 32, 32, 3)
Training labels shape: (45000, 1)

Test images shape: (10000, 32, 32, 3)
Test labels shape: (10000, 1)

Validation images shape: (5000, 32, 32, 3)
Validation labels shape: (5000, 1)
```

Data Preview

In [6]:
```
class_names = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer',
               'Dog', 'Frog', 'Horse', 'Ship', 'Truck']

# Create a new figure
plt.figure(figsize=(15,15))

# Loop over the first 25 images
for i in range(24):
    # Create a subplot for each image
    plt.subplot(8, 8, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)

    # Display the image
    plt.imshow(X_train[i])

    # Set the label as the title
    plt.title(class_names[y_train[i][0]], fontsize=12)

# Display the figure
plt.show()
```

Preprocess the data

Normalisasi Data with Simple Feature Scaling Method

Normalisasi pixel dari data agar menjadi rentang 0 sampai 1 dengan Simple Feature Scaling method. karena nilai pixel disini citra memiliki value dengan rentang 0 sampai 255. di setiap channel (red, green, dan blue) di RGB mode. Semakin mendekati 0 maka semkain gelap (hitam), dan semakin mendekati 255 semakin tinggi intensitas (putih). Normalisasi ini membantu mencapai stabilitas numerik selama pelatihan model dan memastikan bahwa semua fitur berkontribusi sama pada proses pembelajaran. Selain itu, hal ini dapat meningkatkan konvergensi algoritme pengoptimalan dan membuat model kurang sensitif terhadap skala fitur masukan. Namun pada konsep ini, kita harus hati-hati dengan adanya outlier karena outlier dapat mengubah rentang standart dari data

In [7]:
```
# Preprocess the data (e.g., normalize pixel values)
X_train_SFS = X_train / 255.0
X_test_SFS = X_test / 255.0
X_valid_SFS = X_test / 255.0
```

Normalisasi Data with Z-score Method

Normalisasi menggunakan Z-score juga dapat dilakukan terhadap data, Dalam Z-score Scaling, setiap nilai atribut dikurangi dengan rata-rata atribut dan kemudian dibagi dengan standar deviasi atribut. Ini menghasilkan data yang memiliki rata-rata nol dan standar deviasi satu. Teknik ini lebih stabil terhadap outlier karena tidak terpengaruh oleh nilai maksimum atau minimum. Model yang dilatih dengan data yang dinormalisasi dengan teknik ini mungkin lebih sensitif terhadap perbedaan dalam skala atribut.

Dalam Z-score kita juga bisa merepresentasikan dengan mudah nilai datanya, karena jika data bernilai 0 menunjukkan nilai rata-rata dari semua piksel, nilai negatif menunjukkan piksel yang lebih gelap dari rata-rata, dan nilai positif menunjukkan piksel yang lebih terang dari rata-rata.

In [8]:
```
# Calculate the mean and standard deviation of the training images
mean = np.mean(X_train)
std  = np.std(X_train)

# Normalize the data
# The tiny value 1e-7 is added to prevent division by zero
X_train_Zscore = (X_train-mean)/(std+1e-7)
X_test_Zscore  = (X_test-mean) /(std+1e-7)
X_valid_Zscore = (X_valid-mean)/(std+1e-7)
```

One Hot Encoding of Labels

Penggunaan one-hot encoding pada label kelas dalam Convolutional Neural Network (CNN) untuk multi-class classification adalah penting karena beberapa alasan. Pertama, CNN memerlukan representasi kelas dalam bentuk numerik, dan one-hot encoding memberikan cara yang jelas dan langsung untuk mewakili kelas, contohnya jika [0,0,1] berarti Gambar masuk ke kelas nomor 3. Kedua, dengan menggunakan one-hot encoding, output dari CNN memiliki interpretasi yang jelas, memungkinkan kita untuk dengan mudah memahami prediksi model. Terakhir, loss_function yang umum digunakan dalam CNN untuk klasifikasi multi-kelas memerlukan representasi label dalam bentuk one-hot encoding untuk menghitung loss dengan benar.

In [9]:
```
y_train = to_categorical(y_train, 10)
y_valid = to_categorical(y_valid, 10)
y_test  = to_categorical(y_test, 10)
```

Data Augmentation

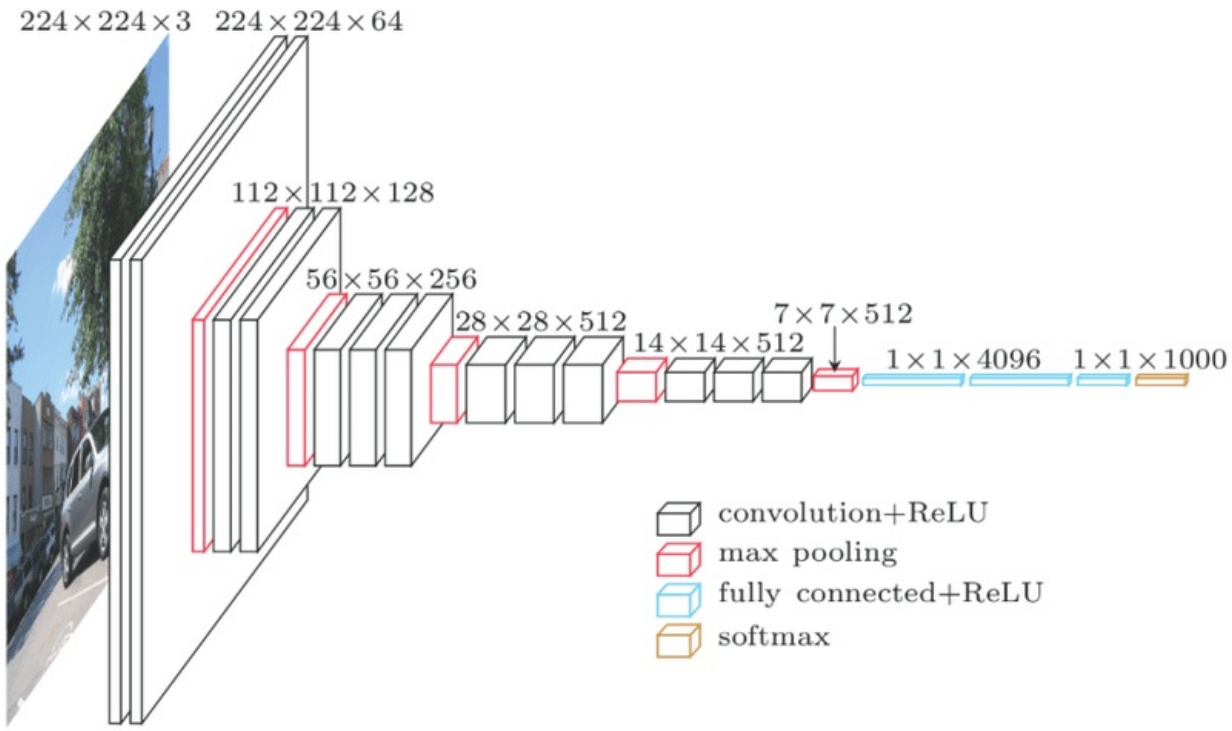
Data Augmentation biasanya digunakan memperluas ukuran set pelatihan secara artifisial dengan membuat versi gambar yang dimodifikasi dalam set data. Hal ini dapat membantu kinerja model agar dapat meningkatkan generalisasi, sehingga mengurangi overfitting. Namun perlu diingat pilihan teknik augmentasi data sering kali bergantung pada karakteristik spesifik kumpulan data dan masalah yang dihadapi.

Data Augmentation digunakan sebagai langkah preprocessing gambar yang berjalan seiring dengan setiap epoch yang ditetapkan. Transformasi gambar ini akan bervariasi pada setiap epoch, menghasilkan variasi yang berbeda dari gambar aslinya. Perubahan ini dilakukan secara dinamis (On-the-Fly) dan tidak mengubah dataset aslinya.

In [10]:
```
# Data augmentation
data_generator = ImageDataGenerator(
    # Rotate images randomly by up to 15 degrees
    rotation_range=15,
    # Shift images horizontally by up to 12% of their width
    width_shift_range=0.12,
    # Shift images vertically by up to 12% of their height
    height_shift_range=0.12,
    # Randomly flip images horizontally
    horizontal_flip=True,
    # Zoom images in by up to 10%
    zoom_range=0.1,
    # Change brightness by up to 10%
    brightness_range=[0.9,1.1],
    # Shear intensity (shear angle in counter-clockwise direction in degrees)
    shear_range=10,
    # Channel shift intensity
    channel_shift_range=0.1,
    # Way to fill pixels after shifting or rotating. 'nearest' is used to fill empty pixels with the closest pi
    fill_mode='nearest'
)
```

Define CNN Model Architecture

VGG16 Network For Inspiration the Architecture model we used now



Arsitektur model terinspirasi oleh **jaringan VGG16**, yang menampilkan beberapa **Convolution Layer** diikuti oleh lapisan **Max_Pooling** dan lapisan **Dropout**, serta lapisan yang terhubung sepenuhnya untuk **klasifikasi**. Ini dimulai dengan memasangkan **lapisan Conv2D** dengan **32 filter** ukuran **3x3**, diikuti oleh **Batch Normalization** untuk **regularisasi** dan mempercepat pelatihan serta membantu mencegah **overfitting**. Lapisan **MaxPooling2D** mengurangi dimensi spasial, diikuti oleh lapisan **Dropout** untuk mencegah overfitting. Pola ini berulang dengan meningkatnya **filter (32, 64, 128, 256)** dan tingkat **dropout (0,2 hingga 0,5)**. Setelah **lapisan konvolusional dan Pooling**, lapisan Ratakan mengubah keluaran menjadi **vektor 1D**, diikuti oleh **Dense (or fully connected) layer** dengan 10 unit untuk klasifikasi menggunakan **aktivasi softmax**. Teknik regularisasi seperti regularisasi L2, Dropout, dan **Batch Normalization**

digunakan untuk efisiensi dan kesederhanaan, dengan fokus pada pembelajaran fitur hierarki dari gambar **CIFAR-10** sekaligus mencegah overfitting. Namun meskipun terinspirasi oleh **VGG16**, model ini tetap sederhana dan tidak menggabungkan fitur-fitur canggih dari arsitektur terkini, melainkan berfokus pada **efisiensi dan kesederhanaan.**

In [11]:
```
# Initialize a sequential model
model = Sequential()

# Set the weight decay value for L2 regularization
weight_decay = 0.0001

# Add the first convolutional layer with 32 filters of size 3x3
model.add(Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu', kernel_regularizer=l2(weight_
                 input_shape=X_train.shape[1:]))
# Add batch normalization layer
model.add(BatchNormalization())

# Add the second convolutional layer similar to the first
model.add(Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu', kernel_regularizer=l2(weight_
model.add(BatchNormalization())

# Add the first max pooling layer with pool size of 2x2
model.add(MaxPooling2D(pool_size=(2, 2)))
# Add dropout layer with 0.2 dropout rate
model.add(Dropout(rate=0.2))

# Add the third and fourth convolutional layers with 64 filters
model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu', kernel_regularizer=l2(weight_
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu', kernel_regularizer=l2(weight_
model.add(BatchNormalization())

# Add the second max pooling layer and increase dropout rate to 0.3
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.3))

# Add the fifth and sixth convolutional layers with 128 filters
model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', activation='relu', kernel_regularizer=l2(weigh
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', activation='relu', kernel_regularizer=l2(weigh
model.add(BatchNormalization())

# Add the third max pooling layer and increase dropout rate to 0.4
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.4))

# Add the seventh and eighth convolutional layers with 256 filters
model.add(Conv2D(filters=256, kernel_size=(3,3), padding='same', activation='relu', kernel_regularizer=l2(weigh
model.add(BatchNormalization())
model.add(Conv2D(filters=256, kernel_size=(3,3), padding='same', activation='relu', kernel_regularizer=l2(weigh
model.add(BatchNormalization())

# Add the fourth max pooling layer and increase dropout rate to 0.5
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.5))

# Flatten the tensor output from the previous layer
model.add(Flatten())

# Add a fully connected layer with softmax activation function for outputting class probabilities
model.add(Dense(10, activation='softmax'))
```

Visualize the CNN Architecture Model

In [12]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (Batch  (None, 32, 32, 32)        128
 Normalization)

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

 batch_normalization_1 (Bat  (None, 32, 32, 32)        128
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 dropout (Dropout)           (None, 16, 16, 32)        0

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

 batch_normalization_2 (Bat  (None, 16, 16, 64)        256
 chNormalization)

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

 batch_normalization_3 (Bat  (None, 16, 16, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

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

 batch_normalization_4 (Bat  (None, 8, 8, 128)         512
 chNormalization)

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

 batch_normalization_5 (Bat  (None, 8, 8, 128)         512
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 dropout_2 (Dropout)         (None, 4, 4, 128)         0

 conv2d_6 (Conv2D)           (None, 4, 4, 256)         295168

 batch_normalization_6 (Bat  (None, 4, 4, 256)         1024
 chNormalization)

 conv2d_7 (Conv2D)           (None, 4, 4, 256)         590080

 batch_normalization_7 (Bat  (None, 4, 4, 256)         1024
 chNormalization)

 max_pooling2d_3 (MaxPoolin  (None, 2, 2, 256)         0
 g2D)

 dropout_3 (Dropout)         (None, 2, 2, 256)         0

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 10)                10250

=================================================================
Total params: 1186346 (4.53 MB)
Trainable params: 1184426 (4.52 MB)
Non-trainable params: 1920 (7.50 KB)
_________________________________________________________________
```

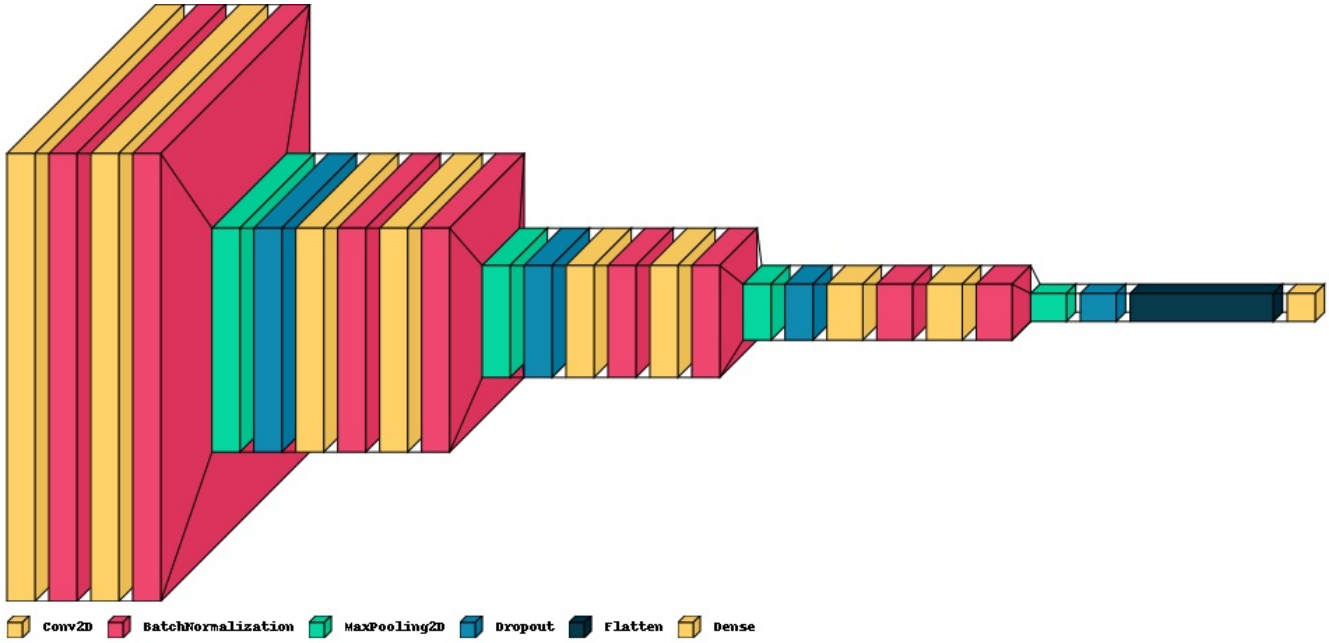
Model kita hanya terdiri dari 1.186.346 parameter, 1.184.426 di antaranya dapat dilatih. Ini adalah model yang relatif kompak, terutama jika dibandingkan dengan arsitektur canggih yang sering kali memiliki puluhan atau bahkan ratusan juta parameter.

Parameter-parameter yang "trainable" dalam sebuah model jaringan saraf adalah parameter-parameter yang disesuaikan selama proses pelatihan untuk meminimalkan fungsi kerugian dan meningkatkan kinerja model. Ini termasuk bobot (weights) dari lapisan-lapisan Conv2D dan Dense, bias, parameter-parameter BatchNormalization yang diatur sebagai trainable, dan parameter-parameter regularisasi. Di sisi lain, parameter-parameter yang "tidak trainable" biasanya terkait dengan lapisan-lapisan khusus seperti lapisan BatchNormalization yang mungkin memiliki parameter "scale" dan "center" yang tidak diatur sebagai trainable untuk tujuan tertentu, atau

parameter-parameter yang disetel statis seperti dalam beberapa konfigurasi praproses data atau lapisan-lapisan khusus yang tidak memerlukan penyesuaian selama pelatihan.

In [13]:
```
visualkeras.layered_view(model, scale_xy=10, legend=True)
```

Out[13]:



Training CNN Model

Sekarang adalah proses pelatihan model jaringan saraf. Proses pelatihan ini menggunakan ukuran batch sebesar 64 dan akan berjalan maksimal selama 250 epoch atau hingga kriteria EarlyStopping terpenuhi. Selama pelatihan, kinerja model dievaluasi pada data validasi setelah setiap epoch. Untuk mengoptimalkan proses pelatihan, saya telah menyertakan dua callback function:

1. Fungsi panggilan ReduceLROnPlateau menyesuaikan learning rate secara dinamis, dengan mengurangi separuhnya (faktor=0.5) ketika loss_validation tidak mengalami perbaikan selama 10 epoch berturut-turut. Penyesuaian ini membantu model untuk mendekati nilai minimum global dari loss_function saat kemajuan terhenti, yang dapat meningkatkan konvergensi pelatihan.
2. Fungsi panggilan EarlyStopping memantau loss_validation dan menghentikan proses pelatihan jika tidak ada peningkatan selama jumlah epoch yang telah ditentukan. Hal ini mencegah penggunaan sumber daya dan waktu yang tidak perlu. Selain itu, fungsi panggilan ini mengembalikan bobot terbaik yang diperoleh selama pelatihan, sehingga kami dapat mempertahankan konfigurasi model yang optimal.

In [14]:
```
# Set the batch size for the training
batch_size = 64

# Set the maximum number of epochs for the training
epochs = 250

# Define the optimizer (Adam)
optimizer = Adam(learning_rate=0.0005)

# Compile the model with the defined optimizer, loss function, and metrics
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# Add ReduceLROnPlateau callback
# Here, the learning rate will be reduced by half (factor=0.5) if no improvement in validation loss is observed
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=10, min_lr=0.00001)

# Add EarlyStopping callback
# Here, training will be stopped if no improvement in validation loss is observed for 40 epochs.
# The `restore_best_weights` parameter ensures that the model weights are reset to the values from the epoch
# with the best value of the monitored quantity (in this case, 'val_loss').
early_stopping = EarlyStopping(monitor='val_loss', patience=40, restore_best_weights=True, verbose=1)

# Fit the model on the training data, using the defined batch size and number of epochs
# The validation data is used to evaluate the model's performance during training
# The callbacks implemented are learning rate reduction when a plateau is reached in validation loss and
# stopping training early if no improvement is observed
model.fit(data_generator.flow(X_train_Zscore, y_train, batch_size=batch_size),
          epochs=epochs,
          validation_data=(X_valid_Zscore, y_valid),
          callbacks=[reduce_lr, early_stopping],
          verbose=2)
```

```
Epoch 1/250
704/704 - 65s - loss: 2.3528 - accuracy: 0.3351 - val_loss: 2.0318 - val_accuracy: 0.3754 - lr: 5.0000e-04 - 65s
/epoch - 92ms/step
Epoch 2/250
704/704 - 50s - loss: 1.6966 - accuracy: 0.4672 - val_loss: 1.8154 - val_accuracy: 0.5286 - lr: 5.0000e-04 - 50s
/epoch - 71ms/step
Epoch 3/250
704/704 - 49s - loss: 1.4673 - accuracy: 0.5321 - val_loss: 1.3484 - val_accuracy: 0.5740 - lr: 5.0000e-04 - 49s
/epoch - 69ms/step
Epoch 4/250
704/704 - 48s - loss: 1.3158 - accuracy: 0.5774 - val_loss: 1.0684 - val_accuracy: 0.6624 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 5/250
704/704 - 49s - loss: 1.1947 - accuracy: 0.6203 - val_loss: 1.0903 - val_accuracy: 0.6646 - lr: 5.0000e-04 - 49s
/epoch - 69ms/step
Epoch 6/250
704/704 - 49s - loss: 1.1249 - accuracy: 0.6450 - val_loss: 1.0919 - val_accuracy: 0.6634 - lr: 5.0000e-04 - 49s
/epoch - 70ms/step
Epoch 7/250
704/704 - 50s - loss: 1.0550 - accuracy: 0.6744 - val_loss: 0.9293 - val_accuracy: 0.7180 - lr: 5.0000e-04 - 50s
/epoch - 71ms/step
Epoch 8/250
704/704 - 49s - loss: 0.9996 - accuracy: 0.6939 - val_loss: 0.9395 - val_accuracy: 0.7126 - lr: 5.0000e-04 - 49s
/epoch - 70ms/step
Epoch 9/250
704/704 - 49s - loss: 0.9798 - accuracy: 0.7050 - val_loss: 0.8697 - val_accuracy: 0.7400 - lr: 5.0000e-04 - 49s
/epoch - 70ms/step
Epoch 10/250
704/704 - 49s - loss: 0.9491 - accuracy: 0.7174 - val_loss: 0.8323 - val_accuracy: 0.7580 - lr: 5.0000e-04 - 49s
/epoch - 69ms/step
Epoch 11/250
704/704 - 48s - loss: 0.9185 - accuracy: 0.7316 - val_loss: 0.7778 - val_accuracy: 0.7704 - lr: 5.0000e-04 - 48s
/epoch - 69ms/step
Epoch 12/250
704/704 - 48s - loss: 0.9135 - accuracy: 0.7373 - val_loss: 0.8573 - val_accuracy: 0.7474 - lr: 5.0000e-04 - 48s
/epoch - 69ms/step
Epoch 13/250
704/704 - 49s - loss: 0.8886 - accuracy: 0.7449 - val_loss: 0.9340 - val_accuracy: 0.7404 - lr: 5.0000e-04 - 49s
/epoch - 70ms/step
Epoch 14/250
704/704 - 49s - loss: 0.8743 - accuracy: 0.7530 - val_loss: 0.7603 - val_accuracy: 0.7934 - lr: 5.0000e-04 - 49s
/epoch - 69ms/step
Epoch 15/250
704/704 - 49s - loss: 0.8562 - accuracy: 0.7607 - val_loss: 0.7026 - val_accuracy: 0.8136 - lr: 5.0000e-04 - 49s
/epoch - 69ms/step
Epoch 16/250
704/704 - 48s - loss: 0.8354 - accuracy: 0.7699 - val_loss: 0.7268 - val_accuracy: 0.8034 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 17/250
704/704 - 48s - loss: 0.8374 - accuracy: 0.7692 - val_loss: 0.8184 - val_accuracy: 0.7822 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 18/250
704/704 - 48s - loss: 0.8350 - accuracy: 0.7742 - val_loss: 0.7354 - val_accuracy: 0.8108 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 19/250
704/704 - 47s - loss: 0.8113 - accuracy: 0.7801 - val_loss: 0.7399 - val_accuracy: 0.8022 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 20/250
704/704 - 47s - loss: 0.8083 - accuracy: 0.7836 - val_loss: 0.7134 - val_accuracy: 0.8206 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 21/250
704/704 - 48s - loss: 0.7793 - accuracy: 0.7932 - val_loss: 0.7673 - val_accuracy: 0.8032 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 22/250
704/704 - 48s - loss: 0.7938 - accuracy: 0.7892 - val_loss: 0.7611 - val_accuracy: 0.8088 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 23/250
704/704 - 48s - loss: 0.7864 - accuracy: 0.7936 - val_loss: 0.6723 - val_accuracy: 0.8386 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 24/250
704/704 - 48s - loss: 0.7818 - accuracy: 0.7953 - val_loss: 0.7133 - val_accuracy: 0.8302 - lr: 5.0000e-04 - 48s
/epoch - 69ms/step
Epoch 25/250
704/704 - 48s - loss: 0.7636 - accuracy: 0.8038 - val_loss: 0.6764 - val_accuracy: 0.8402 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 26/250
704/704 - 49s - loss: 0.7637 - accuracy: 0.8049 - val_loss: 0.6812 - val_accuracy: 0.8356 - lr: 5.0000e-04 - 49s
/epoch - 70ms/step
Epoch 27/250
704/704 - 48s - loss: 0.7530 - accuracy: 0.8072 - val_loss: 0.7101 - val_accuracy: 0.8280 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 28/250
704/704 - 48s - loss: 0.7465 - accuracy: 0.8109 - val_loss: 0.6558 - val_accuracy: 0.8516 - lr: 5.0000e-04 - 48s
```

```
/epoch - 68ms/step
Epoch 29/250
704/704 - 48s - loss: 0.7467 - accuracy: 0.8120 - val_loss: 0.6610 - val_accuracy: 0.8446 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 30/250
704/704 - 47s - loss: 0.7508 - accuracy: 0.8097 - val_loss: 0.6233 - val_accuracy: 0.8568 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 31/250
704/704 - 46s - loss: 0.7443 - accuracy: 0.8136 - val_loss: 0.6639 - val_accuracy: 0.8454 - lr: 5.0000e-04 - 46s
/epoch - 66ms/step
Epoch 32/250
704/704 - 47s - loss: 0.7349 - accuracy: 0.8180 - val_loss: 0.6410 - val_accuracy: 0.8490 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 33/250
704/704 - 48s - loss: 0.7232 - accuracy: 0.8223 - val_loss: 0.6296 - val_accuracy: 0.8550 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 34/250
704/704 - 48s - loss: 0.7226 - accuracy: 0.8215 - val_loss: 0.6506 - val_accuracy: 0.8492 - lr: 5.0000e-04 - 48s
/epoch - 69ms/step
Epoch 35/250
704/704 - 48s - loss: 0.7198 - accuracy: 0.8224 - val_loss: 0.6551 - val_accuracy: 0.8462 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 36/250
704/704 - 48s - loss: 0.7231 - accuracy: 0.8241 - val_loss: 0.6529 - val_accuracy: 0.8498 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 37/250
704/704 - 47s - loss: 0.7198 - accuracy: 0.8233 - val_loss: 0.6092 - val_accuracy: 0.8624 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 38/250
704/704 - 47s - loss: 0.7092 - accuracy: 0.8280 - val_loss: 0.6297 - val_accuracy: 0.8596 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 39/250
704/704 - 47s - loss: 0.7051 - accuracy: 0.8299 - val_loss: 0.6497 - val_accuracy: 0.8572 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 40/250
704/704 - 47s - loss: 0.7108 - accuracy: 0.8274 - val_loss: 0.6437 - val_accuracy: 0.8572 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 41/250
704/704 - 47s - loss: 0.7018 - accuracy: 0.8301 - val_loss: 0.6237 - val_accuracy: 0.8634 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 42/250
704/704 - 47s - loss: 0.7006 - accuracy: 0.8307 - val_loss: 0.6206 - val_accuracy: 0.8636 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 43/250
704/704 - 47s - loss: 0.6975 - accuracy: 0.8316 - val_loss: 0.6126 - val_accuracy: 0.8638 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 44/250
704/704 - 48s - loss: 0.6927 - accuracy: 0.8338 - val_loss: 0.6345 - val_accuracy: 0.8612 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 45/250
704/704 - 48s - loss: 0.6925 - accuracy: 0.8346 - val_loss: 0.6061 - val_accuracy: 0.8642 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 46/250
704/704 - 47s - loss: 0.6903 - accuracy: 0.8340 - val_loss: 0.6284 - val_accuracy: 0.8628 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 47/250
704/704 - 47s - loss: 0.6835 - accuracy: 0.8393 - val_loss: 0.6069 - val_accuracy: 0.8656 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 48/250
704/704 - 48s - loss: 0.6825 - accuracy: 0.8386 - val_loss: 0.6268 - val_accuracy: 0.8614 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 49/250
704/704 - 52s - loss: 0.6777 - accuracy: 0.8394 - val_loss: 0.6096 - val_accuracy: 0.8722 - lr: 5.0000e-04 - 52s
/epoch - 73ms/step
Epoch 50/250
704/704 - 48s - loss: 0.6751 - accuracy: 0.8434 - val_loss: 0.5964 - val_accuracy: 0.8740 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 51/250
704/704 - 47s - loss: 0.6728 - accuracy: 0.8424 - val_loss: 0.6270 - val_accuracy: 0.8564 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 52/250
704/704 - 47s - loss: 0.6647 - accuracy: 0.8455 - val_loss: 0.6214 - val_accuracy: 0.8624 - lr: 5.0000e-04 - 47s
/epoch - 67ms/step
Epoch 53/250
704/704 - 48s - loss: 0.6656 - accuracy: 0.8449 - val_loss: 0.6439 - val_accuracy: 0.8596 - lr: 5.0000e-04 - 48s
/epoch - 69ms/step
Epoch 54/250
704/704 - 48s - loss: 0.6660 - accuracy: 0.8459 - val_loss: 0.6092 - val_accuracy: 0.8684 - lr: 5.0000e-04 - 48s
/epoch - 69ms/step
Epoch 55/250
704/704 - 48s - loss: 0.6655 - accuracy: 0.8461 - val_loss: 0.6290 - val_accuracy: 0.8652 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 56/250
```

```
704/704 - 48s - loss: 0.6644 - accuracy: 0.8454 - val_loss: 0.6065 - val_accuracy: 0.8692 - lr: 5.0000e-04 - 48s
/epoch - 69ms/step
Epoch 57/250
704/704 - 48s - loss: 0.6607 - accuracy: 0.8492 - val_loss: 0.6488 - val_accuracy: 0.8542 - lr: 5.0000e-04 - 48s
/epoch - 69ms/step
Epoch 58/250
704/704 - 48s - loss: 0.6549 - accuracy: 0.8499 - val_loss: 0.6064 - val_accuracy: 0.8718 - lr: 5.0000e-04 - 48s
/epoch - 68ms/step
Epoch 59/250
704/704 - 49s - loss: 0.6576 - accuracy: 0.8493 - val_loss: 0.6144 - val_accuracy: 0.8658 - lr: 5.0000e-04 - 49s
/epoch - 70ms/step
Epoch 60/250
704/704 - 49s - loss: 0.6521 - accuracy: 0.8495 - val_loss: 0.6114 - val_accuracy: 0.8694 - lr: 5.0000e-04 - 49s
/epoch - 70ms/step
Epoch 61/250
704/704 - 50s - loss: 0.6226 - accuracy: 0.8586 - val_loss: 0.5503 - val_accuracy: 0.8844 - lr: 2.5000e-04 - 50s
/epoch - 72ms/step
Epoch 62/250
704/704 - 49s - loss: 0.5950 - accuracy: 0.8661 - val_loss: 0.5578 - val_accuracy: 0.8822 - lr: 2.5000e-04 - 49s
/epoch - 69ms/step
Epoch 63/250
704/704 - 48s - loss: 0.5817 - accuracy: 0.8692 - val_loss: 0.5550 - val_accuracy: 0.8834 - lr: 2.5000e-04 - 48s
/epoch - 69ms/step
Epoch 64/250
704/704 - 48s - loss: 0.5741 - accuracy: 0.8697 - val_loss: 0.5574 - val_accuracy: 0.8808 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 65/250
704/704 - 47s - loss: 0.5698 - accuracy: 0.8721 - val_loss: 0.5312 - val_accuracy: 0.8886 - lr: 2.5000e-04 - 47s
/epoch - 67ms/step
Epoch 66/250
704/704 - 48s - loss: 0.5663 - accuracy: 0.8731 - val_loss: 0.5235 - val_accuracy: 0.8916 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 67/250
704/704 - 48s - loss: 0.5564 - accuracy: 0.8726 - val_loss: 0.5411 - val_accuracy: 0.8848 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 68/250
704/704 - 48s - loss: 0.5528 - accuracy: 0.8736 - val_loss: 0.5108 - val_accuracy: 0.8970 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 69/250
704/704 - 48s - loss: 0.5518 - accuracy: 0.8744 - val_loss: 0.5366 - val_accuracy: 0.8836 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 70/250
704/704 - 49s - loss: 0.5501 - accuracy: 0.8744 - val_loss: 0.5239 - val_accuracy: 0.8928 - lr: 2.5000e-04 - 49s
/epoch - 69ms/step
Epoch 71/250
704/704 - 49s - loss: 0.5429 - accuracy: 0.8758 - val_loss: 0.5111 - val_accuracy: 0.8930 - lr: 2.5000e-04 - 49s
/epoch - 70ms/step
Epoch 72/250
704/704 - 48s - loss: 0.5363 - accuracy: 0.8772 - val_loss: 0.5275 - val_accuracy: 0.8874 - lr: 2.5000e-04 - 48s
/epoch - 69ms/step
Epoch 73/250
704/704 - 48s - loss: 0.5334 - accuracy: 0.8781 - val_loss: 0.5301 - val_accuracy: 0.8894 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 74/250
704/704 - 48s - loss: 0.5321 - accuracy: 0.8776 - val_loss: 0.5515 - val_accuracy: 0.8756 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 75/250
704/704 - 48s - loss: 0.5292 - accuracy: 0.8764 - val_loss: 0.5337 - val_accuracy: 0.8846 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 76/250
704/704 - 48s - loss: 0.5307 - accuracy: 0.8772 - val_loss: 0.5067 - val_accuracy: 0.8952 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 77/250
704/704 - 48s - loss: 0.5240 - accuracy: 0.8806 - val_loss: 0.5261 - val_accuracy: 0.8848 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 78/250
704/704 - 49s - loss: 0.5273 - accuracy: 0.8772 - val_loss: 0.5011 - val_accuracy: 0.8934 - lr: 2.5000e-04 - 49s
/epoch - 69ms/step
Epoch 79/250
704/704 - 48s - loss: 0.5214 - accuracy: 0.8798 - val_loss: 0.5225 - val_accuracy: 0.8846 - lr: 2.5000e-04 - 48s
/epoch - 69ms/step
Epoch 80/250
704/704 - 49s - loss: 0.5196 - accuracy: 0.8798 - val_loss: 0.5280 - val_accuracy: 0.8816 - lr: 2.5000e-04 - 49s
/epoch - 69ms/step
Epoch 81/250
704/704 - 49s - loss: 0.5145 - accuracy: 0.8812 - val_loss: 0.5388 - val_accuracy: 0.8846 - lr: 2.5000e-04 - 49s
/epoch - 69ms/step
Epoch 82/250
704/704 - 48s - loss: 0.5160 - accuracy: 0.8798 - val_loss: 0.5115 - val_accuracy: 0.8922 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 83/250
704/704 - 48s - loss: 0.5202 - accuracy: 0.8775 - val_loss: 0.5236 - val_accuracy: 0.8850 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
```

```
Epoch 84/250
704/704 - 48s - loss: 0.5129 - accuracy: 0.8781 - val_loss: 0.5194 - val_accuracy: 0.8878 - lr: 2.5000e-04 - 48s
/epoch - 69ms/step
Epoch 85/250
704/704 - 49s - loss: 0.5131 - accuracy: 0.8814 - val_loss: 0.4847 - val_accuracy: 0.8986 - lr: 2.5000e-04 - 49s
/epoch - 69ms/step
Epoch 86/250
704/704 - 49s - loss: 0.5085 - accuracy: 0.8818 - val_loss: 0.5128 - val_accuracy: 0.8896 - lr: 2.5000e-04 - 49s
/epoch - 69ms/step
Epoch 87/250
704/704 - 48s - loss: 0.5068 - accuracy: 0.8824 - val_loss: 0.4969 - val_accuracy: 0.8928 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 88/250
704/704 - 47s - loss: 0.5063 - accuracy: 0.8833 - val_loss: 0.4953 - val_accuracy: 0.8900 - lr: 2.5000e-04 - 47s
/epoch - 67ms/step
Epoch 89/250
704/704 - 48s - loss: 0.5075 - accuracy: 0.8838 - val_loss: 0.4963 - val_accuracy: 0.8928 - lr: 2.5000e-04 - 48s
/epoch - 69ms/step
Epoch 90/250
704/704 - 48s - loss: 0.5072 - accuracy: 0.8820 - val_loss: 0.4974 - val_accuracy: 0.8936 - lr: 2.5000e-04 - 48s
/epoch - 69ms/step
Epoch 91/250
704/704 - 48s - loss: 0.5056 - accuracy: 0.8823 - val_loss: 0.4731 - val_accuracy: 0.8996 - lr: 2.5000e-04 - 48s
/epoch - 69ms/step
Epoch 92/250
704/704 - 48s - loss: 0.5025 - accuracy: 0.8831 - val_loss: 0.4866 - val_accuracy: 0.8998 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 93/250
704/704 - 48s - loss: 0.4975 - accuracy: 0.8845 - val_loss: 0.4933 - val_accuracy: 0.8954 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 94/250
704/704 - 49s - loss: 0.5006 - accuracy: 0.8840 - val_loss: 0.4771 - val_accuracy: 0.9000 - lr: 2.5000e-04 - 49s
/epoch - 69ms/step
Epoch 95/250
704/704 - 49s - loss: 0.4961 - accuracy: 0.8828 - val_loss: 0.4904 - val_accuracy: 0.8930 - lr: 2.5000e-04 - 49s
/epoch - 70ms/step
Epoch 96/250
704/704 - 48s - loss: 0.4968 - accuracy: 0.8853 - val_loss: 0.4960 - val_accuracy: 0.8950 - lr: 2.5000e-04 - 48s
/epoch - 69ms/step
Epoch 97/250
704/704 - 48s - loss: 0.4939 - accuracy: 0.8863 - val_loss: 0.5087 - val_accuracy: 0.8866 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 98/250
704/704 - 48s - loss: 0.4988 - accuracy: 0.8823 - val_loss: 0.4732 - val_accuracy: 0.8950 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 99/250
704/704 - 48s - loss: 0.4944 - accuracy: 0.8860 - val_loss: 0.4898 - val_accuracy: 0.8948 - lr: 2.5000e-04 - 48s
/epoch - 68ms/step
Epoch 100/250
704/704 - 46s - loss: 0.4946 - accuracy: 0.8843 - val_loss: 0.5051 - val_accuracy: 0.8892 - lr: 2.5000e-04 - 46s
/epoch - 65ms/step
Epoch 101/250
704/704 - 47s - loss: 0.4958 - accuracy: 0.8834 - val_loss: 0.5357 - val_accuracy: 0.8798 - lr: 2.5000e-04 - 47s
/epoch - 67ms/step
Epoch 102/250
704/704 - 48s - loss: 0.4632 - accuracy: 0.8958 - val_loss: 0.4712 - val_accuracy: 0.8982 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 103/250
704/704 - 47s - loss: 0.4561 - accuracy: 0.8972 - val_loss: 0.4739 - val_accuracy: 0.8974 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 104/250
704/704 - 47s - loss: 0.4496 - accuracy: 0.8989 - val_loss: 0.4826 - val_accuracy: 0.8964 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 105/250
704/704 - 48s - loss: 0.4455 - accuracy: 0.8993 - val_loss: 0.4691 - val_accuracy: 0.9006 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 106/250
704/704 - 48s - loss: 0.4411 - accuracy: 0.9008 - val_loss: 0.4673 - val_accuracy: 0.8986 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 107/250
704/704 - 48s - loss: 0.4445 - accuracy: 0.8985 - val_loss: 0.4698 - val_accuracy: 0.9010 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 108/250
704/704 - 48s - loss: 0.4430 - accuracy: 0.8981 - val_loss: 0.4796 - val_accuracy: 0.8948 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 109/250
704/704 - 47s - loss: 0.4308 - accuracy: 0.9029 - val_loss: 0.4637 - val_accuracy: 0.8998 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 110/250
704/704 - 48s - loss: 0.4368 - accuracy: 0.9015 - val_loss: 0.4549 - val_accuracy: 0.9048 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 111/250
704/704 - 48s - loss: 0.4371 - accuracy: 0.8995 - val_loss: 0.4492 - val_accuracy: 0.9012 - lr: 1.2500e-04 - 48s
```

```
/epoch - 68ms/step
Epoch 112/250
704/704 - 48s - loss: 0.4302 - accuracy: 0.9033 - val_loss: 0.4598 - val_accuracy: 0.8998 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 113/250
704/704 - 48s - loss: 0.4264 - accuracy: 0.9029 - val_loss: 0.4484 - val_accuracy: 0.9038 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 114/250
704/704 - 48s - loss: 0.4292 - accuracy: 0.9016 - val_loss: 0.4516 - val_accuracy: 0.9032 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 115/250
704/704 - 48s - loss: 0.4220 - accuracy: 0.9056 - val_loss: 0.4416 - val_accuracy: 0.9044 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 116/250
704/704 - 48s - loss: 0.4210 - accuracy: 0.9038 - val_loss: 0.4360 - val_accuracy: 0.9100 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 117/250
704/704 - 48s - loss: 0.4154 - accuracy: 0.9049 - val_loss: 0.4463 - val_accuracy: 0.9044 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 118/250
704/704 - 48s - loss: 0.4192 - accuracy: 0.9041 - val_loss: 0.4502 - val_accuracy: 0.9060 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 119/250
704/704 - 48s - loss: 0.4142 - accuracy: 0.9061 - val_loss: 0.4343 - val_accuracy: 0.9098 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 120/250
704/704 - 48s - loss: 0.4191 - accuracy: 0.9046 - val_loss: 0.4524 - val_accuracy: 0.9036 - lr: 1.2500e-04 - 48s
/epoch - 69ms/step
Epoch 121/250
704/704 - 48s - loss: 0.4185 - accuracy: 0.9039 - val_loss: 0.4532 - val_accuracy: 0.9004 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 122/250
704/704 - 48s - loss: 0.4108 - accuracy: 0.9065 - val_loss: 0.4339 - val_accuracy: 0.9076 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 123/250
704/704 - 48s - loss: 0.4139 - accuracy: 0.9072 - val_loss: 0.4502 - val_accuracy: 0.9020 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 124/250
704/704 - 48s - loss: 0.4057 - accuracy: 0.9070 - val_loss: 0.4553 - val_accuracy: 0.9022 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 125/250
704/704 - 48s - loss: 0.4077 - accuracy: 0.9056 - val_loss: 0.4421 - val_accuracy: 0.9038 - lr: 1.2500e-04 - 48s
/epoch - 69ms/step
Epoch 126/250
704/704 - 47s - loss: 0.4108 - accuracy: 0.9046 - val_loss: 0.4343 - val_accuracy: 0.9070 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 127/250
704/704 - 48s - loss: 0.4103 - accuracy: 0.9050 - val_loss: 0.4542 - val_accuracy: 0.9004 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 128/250
704/704 - 47s - loss: 0.4048 - accuracy: 0.9069 - val_loss: 0.4384 - val_accuracy: 0.9070 - lr: 1.2500e-04 - 47s
/epoch - 66ms/step
Epoch 129/250
704/704 - 47s - loss: 0.4001 - accuracy: 0.9074 - val_loss: 0.4296 - val_accuracy: 0.9082 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 130/250
704/704 - 48s - loss: 0.4008 - accuracy: 0.9069 - val_loss: 0.4632 - val_accuracy: 0.8966 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 131/250
704/704 - 48s - loss: 0.4043 - accuracy: 0.9072 - val_loss: 0.4484 - val_accuracy: 0.9026 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 132/250
704/704 - 48s - loss: 0.3961 - accuracy: 0.9089 - val_loss: 0.4478 - val_accuracy: 0.9082 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 133/250
704/704 - 48s - loss: 0.4003 - accuracy: 0.9076 - val_loss: 0.4272 - val_accuracy: 0.9108 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 134/250
704/704 - 48s - loss: 0.3986 - accuracy: 0.9079 - val_loss: 0.4371 - val_accuracy: 0.9042 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 135/250
704/704 - 47s - loss: 0.3942 - accuracy: 0.9088 - val_loss: 0.4401 - val_accuracy: 0.9022 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 136/250
704/704 - 48s - loss: 0.3971 - accuracy: 0.9084 - val_loss: 0.4484 - val_accuracy: 0.9022 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 137/250
704/704 - 47s - loss: 0.4021 - accuracy: 0.9066 - val_loss: 0.4458 - val_accuracy: 0.9002 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 138/250
704/704 - 47s - loss: 0.3962 - accuracy: 0.9079 - val_loss: 0.4488 - val_accuracy: 0.9000 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 139/250
```

```
704/704 - 47s - loss: 0.3921 - accuracy: 0.9094 - val_loss: 0.4308 - val_accuracy: 0.9050 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 140/250
704/704 - 47s - loss: 0.3969 - accuracy: 0.9088 - val_loss: 0.4320 - val_accuracy: 0.9068 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 141/250
704/704 - 47s - loss: 0.3896 - accuracy: 0.9099 - val_loss: 0.4256 - val_accuracy: 0.9078 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 142/250
704/704 - 48s - loss: 0.3943 - accuracy: 0.9095 - val_loss: 0.4209 - val_accuracy: 0.9092 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 143/250
704/704 - 48s - loss: 0.3886 - accuracy: 0.9101 - val_loss: 0.4333 - val_accuracy: 0.9018 - lr: 1.2500e-04 - 48s
/epoch - 69ms/step
Epoch 144/250
704/704 - 47s - loss: 0.3883 - accuracy: 0.9120 - val_loss: 0.4349 - val_accuracy: 0.9072 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 145/250
704/704 - 48s - loss: 0.3887 - accuracy: 0.9104 - val_loss: 0.4187 - val_accuracy: 0.9080 - lr: 1.2500e-04 - 48s
/epoch - 67ms/step
Epoch 146/250
704/704 - 48s - loss: 0.3893 - accuracy: 0.9102 - val_loss: 0.4364 - val_accuracy: 0.9050 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 147/250
704/704 - 55s - loss: 0.3863 - accuracy: 0.9115 - val_loss: 0.4514 - val_accuracy: 0.9002 - lr: 1.2500e-04 - 55s
/epoch - 78ms/step
Epoch 148/250
704/704 - 48s - loss: 0.3816 - accuracy: 0.9123 - val_loss: 0.4460 - val_accuracy: 0.9020 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 149/250
704/704 - 49s - loss: 0.3816 - accuracy: 0.9124 - val_loss: 0.4365 - val_accuracy: 0.8992 - lr: 1.2500e-04 - 49s
/epoch - 69ms/step
Epoch 150/250
704/704 - 47s - loss: 0.3833 - accuracy: 0.9125 - val_loss: 0.4512 - val_accuracy: 0.8962 - lr: 1.2500e-04 - 47s
/epoch - 66ms/step
Epoch 151/250
704/704 - 47s - loss: 0.3833 - accuracy: 0.9110 - val_loss: 0.4626 - val_accuracy: 0.8972 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 152/250
704/704 - 48s - loss: 0.3884 - accuracy: 0.9089 - val_loss: 0.4389 - val_accuracy: 0.9028 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 153/250
704/704 - 48s - loss: 0.3856 - accuracy: 0.9101 - val_loss: 0.4469 - val_accuracy: 0.8970 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 154/250
704/704 - 48s - loss: 0.3835 - accuracy: 0.9107 - val_loss: 0.4142 - val_accuracy: 0.9080 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 155/250
704/704 - 47s - loss: 0.3826 - accuracy: 0.9106 - val_loss: 0.4335 - val_accuracy: 0.9038 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 156/250
704/704 - 48s - loss: 0.3859 - accuracy: 0.9088 - val_loss: 0.4557 - val_accuracy: 0.9012 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 157/250
704/704 - 48s - loss: 0.3837 - accuracy: 0.9112 - val_loss: 0.4358 - val_accuracy: 0.9064 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 158/250
704/704 - 48s - loss: 0.3789 - accuracy: 0.9124 - val_loss: 0.4243 - val_accuracy: 0.9056 - lr: 1.2500e-04 - 48s
/epoch - 68ms/step
Epoch 159/250
704/704 - 47s - loss: 0.3785 - accuracy: 0.9119 - val_loss: 0.4281 - val_accuracy: 0.9018 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 160/250
704/704 - 47s - loss: 0.3776 - accuracy: 0.9125 - val_loss: 0.4380 - val_accuracy: 0.8980 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 161/250
704/704 - 47s - loss: 0.3808 - accuracy: 0.9120 - val_loss: 0.4212 - val_accuracy: 0.9108 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 162/250
704/704 - 47s - loss: 0.3769 - accuracy: 0.9119 - val_loss: 0.4240 - val_accuracy: 0.9074 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 163/250
704/704 - 46s - loss: 0.3757 - accuracy: 0.9114 - val_loss: 0.4402 - val_accuracy: 0.9018 - lr: 1.2500e-04 - 46s
/epoch - 66ms/step
Epoch 164/250
704/704 - 47s - loss: 0.3801 - accuracy: 0.9104 - val_loss: 0.4271 - val_accuracy: 0.9036 - lr: 1.2500e-04 - 47s
/epoch - 67ms/step
Epoch 165/250
704/704 - 48s - loss: 0.3703 - accuracy: 0.9146 - val_loss: 0.4208 - val_accuracy: 0.9064 - lr: 6.2500e-05 - 48s
/epoch - 68ms/step
Epoch 166/250
704/704 - 47s - loss: 0.3568 - accuracy: 0.9190 - val_loss: 0.4174 - val_accuracy: 0.9072 - lr: 6.2500e-05 - 47s
/epoch - 67ms/step
```

```
Epoch 167/250
704/704 - 47s - loss: 0.3537 - accuracy: 0.9202 - val_loss: 0.4120 - val_accuracy: 0.9118 - lr: 6.2500e-05 - 47s
/epoch - 67ms/step
Epoch 168/250
704/704 - 47s - loss: 0.3532 - accuracy: 0.9202 - val_loss: 0.4110 - val_accuracy: 0.9096 - lr: 6.2500e-05 - 47s
/epoch - 67ms/step
Epoch 169/250
704/704 - 48s - loss: 0.3545 - accuracy: 0.9181 - val_loss: 0.4106 - val_accuracy: 0.9092 - lr: 6.2500e-05 - 48s
/epoch - 68ms/step
Epoch 170/250
704/704 - 47s - loss: 0.3525 - accuracy: 0.9198 - val_loss: 0.4124 - val_accuracy: 0.9066 - lr: 6.2500e-05 - 47s
/epoch - 67ms/step
Epoch 171/250
704/704 - 48s - loss: 0.3503 - accuracy: 0.9202 - val_loss: 0.4001 - val_accuracy: 0.9122 - lr: 6.2500e-05 - 48s
/epoch - 68ms/step
Epoch 172/250
704/704 - 48s - loss: 0.3481 - accuracy: 0.9207 - val_loss: 0.4219 - val_accuracy: 0.9080 - lr: 6.2500e-05 - 48s
/epoch - 68ms/step
Epoch 173/250
704/704 - 48s - loss: 0.3471 - accuracy: 0.9220 - val_loss: 0.4149 - val_accuracy: 0.9066 - lr: 6.2500e-05 - 48s
/epoch - 68ms/step
Epoch 174/250
704/704 - 48s - loss: 0.3476 - accuracy: 0.9216 - val_loss: 0.4135 - val_accuracy: 0.9104 - lr: 6.2500e-05 - 48s
/epoch - 68ms/step
Epoch 175/250
704/704 - 48s - loss: 0.3452 - accuracy: 0.9217 - val_loss: 0.4055 - val_accuracy: 0.9110 - lr: 6.2500e-05 - 48s
/epoch - 69ms/step
Epoch 176/250
704/704 - 48s - loss: 0.3445 - accuracy: 0.9213 - val_loss: 0.4288 - val_accuracy: 0.9070 - lr: 6.2500e-05 - 48s
/epoch - 68ms/step
Epoch 177/250
704/704 - 47s - loss: 0.3504 - accuracy: 0.9193 - val_loss: 0.4188 - val_accuracy: 0.9088 - lr: 6.2500e-05 - 47s
/epoch - 67ms/step
Epoch 178/250
704/704 - 47s - loss: 0.3433 - accuracy: 0.9203 - val_loss: 0.4121 - val_accuracy: 0.9110 - lr: 6.2500e-05 - 47s
/epoch - 67ms/step
Epoch 179/250
704/704 - 47s - loss: 0.3434 - accuracy: 0.9213 - val_loss: 0.4136 - val_accuracy: 0.9092 - lr: 6.2500e-05 - 47s
/epoch - 66ms/step
Epoch 180/250
704/704 - 48s - loss: 0.3411 - accuracy: 0.9230 - val_loss: 0.4040 - val_accuracy: 0.9136 - lr: 6.2500e-05 - 48s
/epoch - 68ms/step
Epoch 181/250
704/704 - 48s - loss: 0.3392 - accuracy: 0.9238 - val_loss: 0.4163 - val_accuracy: 0.9084 - lr: 6.2500e-05 - 48s
/epoch - 68ms/step
Epoch 182/250
704/704 - 48s - loss: 0.3327 - accuracy: 0.9244 - val_loss: 0.4085 - val_accuracy: 0.9106 - lr: 3.1250e-05 - 48s
/epoch - 68ms/step
Epoch 183/250
704/704 - 48s - loss: 0.3307 - accuracy: 0.9260 - val_loss: 0.4142 - val_accuracy: 0.9092 - lr: 3.1250e-05 - 48s
/epoch - 68ms/step
Epoch 184/250
704/704 - 48s - loss: 0.3272 - accuracy: 0.9276 - val_loss: 0.4136 - val_accuracy: 0.9114 - lr: 3.1250e-05 - 48s
/epoch - 68ms/step
Epoch 185/250
704/704 - 47s - loss: 0.3294 - accuracy: 0.9260 - val_loss: 0.4112 - val_accuracy: 0.9086 - lr: 3.1250e-05 - 47s
/epoch - 67ms/step
Epoch 186/250
704/704 - 48s - loss: 0.3252 - accuracy: 0.9261 - val_loss: 0.4079 - val_accuracy: 0.9118 - lr: 3.1250e-05 - 48s
/epoch - 68ms/step
Epoch 187/250
704/704 - 47s - loss: 0.3238 - accuracy: 0.9280 - val_loss: 0.4009 - val_accuracy: 0.9126 - lr: 3.1250e-05 - 47s
/epoch - 67ms/step
Epoch 188/250
704/704 - 47s - loss: 0.3244 - accuracy: 0.9272 - val_loss: 0.4109 - val_accuracy: 0.9094 - lr: 3.1250e-05 - 47s
/epoch - 67ms/step
Epoch 189/250
704/704 - 47s - loss: 0.3201 - accuracy: 0.9302 - val_loss: 0.4070 - val_accuracy: 0.9128 - lr: 3.1250e-05 - 47s
/epoch - 67ms/step
Epoch 190/250
704/704 - 47s - loss: 0.3280 - accuracy: 0.9264 - val_loss: 0.4100 - val_accuracy: 0.9090 - lr: 3.1250e-05 - 47s
/epoch - 67ms/step
Epoch 191/250
704/704 - 47s - loss: 0.3284 - accuracy: 0.9260 - val_loss: 0.4050 - val_accuracy: 0.9096 - lr: 3.1250e-05 - 47s
/epoch - 66ms/step
Epoch 192/250
704/704 - 47s - loss: 0.3215 - accuracy: 0.9291 - val_loss: 0.4059 - val_accuracy: 0.9106 - lr: 1.5625e-05 - 47s
/epoch - 67ms/step
Epoch 193/250
704/704 - 46s - loss: 0.3197 - accuracy: 0.9290 - val_loss: 0.3989 - val_accuracy: 0.9112 - lr: 1.5625e-05 - 46s
/epoch - 65ms/step
Epoch 194/250
704/704 - 47s - loss: 0.3149 - accuracy: 0.9312 - val_loss: 0.3978 - val_accuracy: 0.9130 - lr: 1.5625e-05 - 47s
```

```
/epoch - 67ms/step
Epoch 195/250
704/704 - 49s - loss: 0.3194 - accuracy: 0.9288 - val_loss: 0.4017 - val_accuracy: 0.9118 - lr: 1.5625e-05 - 49s
/epoch - 69ms/step
Epoch 196/250
704/704 - 47s - loss: 0.3150 - accuracy: 0.9291 - val_loss: 0.4013 - val_accuracy: 0.9114 - lr: 1.5625e-05 - 47s
/epoch - 67ms/step
Epoch 197/250
704/704 - 47s - loss: 0.3136 - accuracy: 0.9296 - val_loss: 0.3985 - val_accuracy: 0.9130 - lr: 1.5625e-05 - 47s
/epoch - 66ms/step
Epoch 198/250
704/704 - 48s - loss: 0.3139 - accuracy: 0.9296 - val_loss: 0.3987 - val_accuracy: 0.9138 - lr: 1.5625e-05 - 48s
/epoch - 68ms/step
Epoch 199/250
704/704 - 48s - loss: 0.3150 - accuracy: 0.9295 - val_loss: 0.4032 - val_accuracy: 0.9098 - lr: 1.5625e-05 - 48s
/epoch - 68ms/step
Epoch 200/250
704/704 - 48s - loss: 0.3147 - accuracy: 0.9306 - val_loss: 0.3989 - val_accuracy: 0.9128 - lr: 1.5625e-05 - 48s
/epoch - 68ms/step
Epoch 201/250
704/704 - 48s - loss: 0.3100 - accuracy: 0.9313 - val_loss: 0.3982 - val_accuracy: 0.9124 - lr: 1.5625e-05 - 48s
/epoch - 68ms/step
Epoch 202/250
704/704 - 48s - loss: 0.3129 - accuracy: 0.9314 - val_loss: 0.4016 - val_accuracy: 0.9126 - lr: 1.5625e-05 - 48s
/epoch - 69ms/step
Epoch 203/250
704/704 - 48s - loss: 0.3098 - accuracy: 0.9307 - val_loss: 0.3988 - val_accuracy: 0.9142 - lr: 1.5625e-05 - 48s
/epoch - 68ms/step
Epoch 204/250
704/704 - 47s - loss: 0.3145 - accuracy: 0.9294 - val_loss: 0.4016 - val_accuracy: 0.9114 - lr: 1.5625e-05 - 47s
/epoch - 67ms/step
Epoch 205/250
704/704 - 47s - loss: 0.3094 - accuracy: 0.9317 - val_loss: 0.4044 - val_accuracy: 0.9106 - lr: 1.0000e-05 - 47s
/epoch - 66ms/step
Epoch 206/250
704/704 - 47s - loss: 0.3115 - accuracy: 0.9308 - val_loss: 0.3969 - val_accuracy: 0.9112 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 207/250
704/704 - 47s - loss: 0.3116 - accuracy: 0.9312 - val_loss: 0.4003 - val_accuracy: 0.9122 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 208/250
704/704 - 47s - loss: 0.3077 - accuracy: 0.9331 - val_loss: 0.3958 - val_accuracy: 0.9126 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 209/250
704/704 - 47s - loss: 0.3047 - accuracy: 0.9325 - val_loss: 0.4013 - val_accuracy: 0.9116 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 210/250
704/704 - 47s - loss: 0.3096 - accuracy: 0.9322 - val_loss: 0.3971 - val_accuracy: 0.9128 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 211/250
704/704 - 48s - loss: 0.3086 - accuracy: 0.9325 - val_loss: 0.3962 - val_accuracy: 0.9132 - lr: 1.0000e-05 - 48s
/epoch - 69ms/step
Epoch 212/250
704/704 - 47s - loss: 0.3033 - accuracy: 0.9340 - val_loss: 0.3994 - val_accuracy: 0.9112 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 213/250
704/704 - 47s - loss: 0.3093 - accuracy: 0.9323 - val_loss: 0.3998 - val_accuracy: 0.9112 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 214/250
704/704 - 47s - loss: 0.3073 - accuracy: 0.9334 - val_loss: 0.3988 - val_accuracy: 0.9108 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 215/250
704/704 - 47s - loss: 0.3076 - accuracy: 0.9327 - val_loss: 0.3970 - val_accuracy: 0.9146 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 216/250
704/704 - 46s - loss: 0.3064 - accuracy: 0.9327 - val_loss: 0.3967 - val_accuracy: 0.9126 - lr: 1.0000e-05 - 46s
/epoch - 66ms/step
Epoch 217/250
704/704 - 47s - loss: 0.3051 - accuracy: 0.9327 - val_loss: 0.3998 - val_accuracy: 0.9114 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 218/250
704/704 - 47s - loss: 0.3084 - accuracy: 0.9326 - val_loss: 0.3986 - val_accuracy: 0.9120 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 219/250
704/704 - 47s - loss: 0.3051 - accuracy: 0.9322 - val_loss: 0.3970 - val_accuracy: 0.9132 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 220/250
704/704 - 47s - loss: 0.3070 - accuracy: 0.9325 - val_loss: 0.4016 - val_accuracy: 0.9110 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 221/250
704/704 - 48s - loss: 0.3065 - accuracy: 0.9334 - val_loss: 0.3981 - val_accuracy: 0.9116 - lr: 1.0000e-05 - 48s
/epoch - 68ms/step
Epoch 222/250
```

```
704/704 - 46s - loss: 0.3059 - accuracy: 0.9318 - val_loss: 0.3996 - val_accuracy: 0.9132 - lr: 1.0000e-05 - 46s
/epoch - 66ms/step
Epoch 223/250
704/704 - 47s - loss: 0.3027 - accuracy: 0.9328 - val_loss: 0.4004 - val_accuracy: 0.9124 - lr: 1.0000e-05 - 47s
/epoch - 66ms/step
Epoch 224/250
704/704 - 47s - loss: 0.3025 - accuracy: 0.9334 - val_loss: 0.3971 - val_accuracy: 0.9124 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 225/250
704/704 - 47s - loss: 0.3028 - accuracy: 0.9334 - val_loss: 0.3991 - val_accuracy: 0.9114 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 226/250
704/704 - 47s - loss: 0.2994 - accuracy: 0.9341 - val_loss: 0.3962 - val_accuracy: 0.9128 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 227/250
704/704 - 47s - loss: 0.3028 - accuracy: 0.9340 - val_loss: 0.3985 - val_accuracy: 0.9122 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 228/250
704/704 - 46s - loss: 0.3063 - accuracy: 0.9321 - val_loss: 0.4010 - val_accuracy: 0.9128 - lr: 1.0000e-05 - 46s
/epoch - 66ms/step
Epoch 229/250
704/704 - 47s - loss: 0.2996 - accuracy: 0.9346 - val_loss: 0.3974 - val_accuracy: 0.9118 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 230/250
704/704 - 47s - loss: 0.3049 - accuracy: 0.9319 - val_loss: 0.3977 - val_accuracy: 0.9130 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 231/250
704/704 - 47s - loss: 0.3026 - accuracy: 0.9332 - val_loss: 0.3941 - val_accuracy: 0.9136 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 232/250
704/704 - 47s - loss: 0.3021 - accuracy: 0.9332 - val_loss: 0.4005 - val_accuracy: 0.9134 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 233/250
704/704 - 48s - loss: 0.3043 - accuracy: 0.9328 - val_loss: 0.4017 - val_accuracy: 0.9122 - lr: 1.0000e-05 - 48s
/epoch - 68ms/step
Epoch 234/250
704/704 - 47s - loss: 0.3027 - accuracy: 0.9329 - val_loss: 0.3990 - val_accuracy: 0.9128 - lr: 1.0000e-05 - 47s
/epoch - 66ms/step
Epoch 235/250
704/704 - 46s - loss: 0.3009 - accuracy: 0.9333 - val_loss: 0.4033 - val_accuracy: 0.9124 - lr: 1.0000e-05 - 46s
/epoch - 65ms/step
Epoch 236/250
704/704 - 47s - loss: 0.3016 - accuracy: 0.9330 - val_loss: 0.4022 - val_accuracy: 0.9122 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 237/250
704/704 - 47s - loss: 0.2980 - accuracy: 0.9351 - val_loss: 0.4049 - val_accuracy: 0.9114 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 238/250
704/704 - 47s - loss: 0.3000 - accuracy: 0.9334 - val_loss: 0.4001 - val_accuracy: 0.9122 - lr: 1.0000e-05 - 47s
/epoch - 66ms/step
Epoch 239/250
704/704 - 47s - loss: 0.3002 - accuracy: 0.9328 - val_loss: 0.3975 - val_accuracy: 0.9126 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 240/250
704/704 - 47s - loss: 0.3020 - accuracy: 0.9338 - val_loss: 0.4064 - val_accuracy: 0.9108 - lr: 1.0000e-05 - 47s
/epoch - 66ms/step
Epoch 241/250
704/704 - 46s - loss: 0.3025 - accuracy: 0.9332 - val_loss: 0.3988 - val_accuracy: 0.9126 - lr: 1.0000e-05 - 46s
/epoch - 65ms/step
Epoch 242/250
704/704 - 47s - loss: 0.2997 - accuracy: 0.9338 - val_loss: 0.4014 - val_accuracy: 0.9106 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 243/250
704/704 - 47s - loss: 0.2981 - accuracy: 0.9337 - val_loss: 0.4015 - val_accuracy: 0.9122 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 244/250
704/704 - 47s - loss: 0.3009 - accuracy: 0.9337 - val_loss: 0.3996 - val_accuracy: 0.9114 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 245/250
704/704 - 47s - loss: 0.2989 - accuracy: 0.9349 - val_loss: 0.4001 - val_accuracy: 0.9104 - lr: 1.0000e-05 - 47s
/epoch - 66ms/step
Epoch 246/250
704/704 - 47s - loss: 0.3038 - accuracy: 0.9325 - val_loss: 0.3955 - val_accuracy: 0.9136 - lr: 1.0000e-05 - 47s
/epoch - 67ms/step
Epoch 247/250
704/704 - 46s - loss: 0.3013 - accuracy: 0.9340 - val_loss: 0.3968 - val_accuracy: 0.9124 - lr: 1.0000e-05 - 46s
/epoch - 65ms/step
Epoch 248/250
704/704 - 46s - loss: 0.2963 - accuracy: 0.9347 - val_loss: 0.4001 - val_accuracy: 0.9112 - lr: 1.0000e-05 - 46s
/epoch - 66ms/step
Epoch 249/250
704/704 - 49s - loss: 0.2982 - accuracy: 0.9342 - val_loss: 0.4012 - val_accuracy: 0.9110 - lr: 1.0000e-05 - 49s
/epoch - 69ms/step
```

```
Epoch 250/250
704/704 - 48s - loss: 0.2962 - accuracy: 0.9351 - val_loss: 0.4016 - val_accuracy: 0.9132 - lr: 1.0000e-05 - 48s
/epoch - 68ms/step
```

Out[14]:
```
<keras.src.callbacks.History at 0x7e7bed8dc490>
```

In [25]:
```
#  3.472 jam running model - 93% Akurasi Untuk Data Latih
```

Save the model

In [16]:
```
from keras.models import load_model

# Simpan model
model.save('/content/drive/MyDrive/Colab Notebooks/1. Bisa Ai/Natural Language Preprocessing/cifar-10 classifica

# Memuat model
model = load_model('/content/drive/MyDrive/Colab Notebooks/1. Bisa Ai/Natural Language Preprocessing/cifar-10 c
```

Evaluating the Optimal Model on Test Data

Karena kita telah menetapkan recovery_best_weights=True di EarlyStopping, setelah pelatihan, model itu sendiri akan memiliki bobot terbaik. Setelah ini, saya akan menggunakan model ini untuk mengevaluasi kinerjanya pada data pengujian, menghitung akurasi dan kerugian pengujian

In [24]:
```
# Use the model to make predictions, evaluate on test data
test_loss, test_acc = model.evaluate(X_test_Zscore, y_test, verbose=1)

print('\nTest Accuracy:', test_acc)
print('Test Loss:    ', test_loss)
```

```
313/313 [==============================] - 2s 4ms/step - loss: 0.4266 - accuracy: 0.9070

Test Accuracy: 0.9070000052452087
Test Loss:     0.4266330599784851
```

Dengan akurasi pengujian sekitar ***90%***, model ini menunjukkan kinerja yang sangat baik pada data uji yang tidak dikenal sebelumnya. Meskipun memiliki jumlah parameter yang relatif sedikit, hanya sekitar 1,2 juta, model ini berhasil mencapai akurasi yang tinggi. Hal ini menarik karena banyak arsitektur yang lebih kompleks menggunakan jutaan bahkan puluhan juta parameter untuk mencapai hasil serupa atau hanya sedikit lebih baik. Fakta bahwa loss dan akurasi pengujian berada dalam jarak yang dekat dengan model pelatihan menunjukkan bahwa model ini tidak hanya menghafal data pelatihan, tetapi juga memahami pola dengan baik dan mampu menggeneralisasi dari data pelatihan ke data yang belum pernah dilihat sebelumnya.

Performance on an Out-of-Dataset Image

In [52]:
```
import urllib.request
import cv2

# Fetch the raw image from GitHub
url = "https://www.akc.org/wp-content/uploads/2017/11/GettyImages-187066830.jpg"
resp = urllib.request.urlopen(url)
image = np.asarray(bytearray(resp.read()), dtype="uint8")
image = cv2.imdecode(image, cv2.IMREAD_UNCHANGED)

# Convert the image from BGR to RGB
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

In [53]:
```
# Display the image
plt.imshow(image)
plt.xticks([])
plt.yticks([])
plt.grid(False)
plt.show()
```

In [54]:
```
# Resize it to 32x32 pixels
image = cv2.resize(image, (32,32))
```

In [55]:
```
# Calculate the mean and standard deviation of the training images
mean = np.mean(image)
std  = np.std(image)

# Normalize the data
# The tiny value 1e-7 is added to prevent division by zero
image_Zscore = (image-mean)/(std+1e-7)

# Add an extra dimension because the model expects a batch of images
image_Zscore = image_Zscore.reshape((1, 32, 32, 3))
```

In [56]:
```
prediction = model.predict(image_Zscore)
```

```
1/1 [==============================] - 0s 62ms/step
```

In [57]:
```
predicted_class = prediction.argmax()

print('Predicted class: ', class_names[predicted_class])
```

```
Predicted class:  Dog
```