Name: Priyanka Eluri

Student ID: 700756198

Github link: https://github.com/peluri03/NN_IPC_4

In [6]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.constraints import max_norm
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.callbacks import LearningRateScheduler
```

In [7]:
```
np.random.seed(7)
```

In [8]:
```
(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 [==============================] - 2s 0us/step
```

In [9]:
```
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
```

In [10]:
```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
num_classes = y_test.shape[1]
```

In [11]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu', kernel_constraint=max_norm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
```

In [11]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu', kernel_constraint=max_norm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_constraint=max_norm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

In [12]:
```
import tensorflow as tf
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers.schedules import ExponentialDecay
from tensorflow.keras.callbacks import LearningRateScheduler

# Define a learning rate schedule using ExponentialDecay
initial_learning_rate = 0.01
decay_steps = 10000
decay_rate = 0.9
learning_rate_schedule = ExponentialDecay(
    initial_learning_rate, decay_steps, decay_rate, staircase=True
)

# Create the SGD optimizer with the learning rate schedule
sgd = SGD(learning_rate=learning_rate_schedule, momentum=0.9)

# Compile your model
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

# Print the model summary
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

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

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

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

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 512)               4194816

 dropout_1 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 4210090 (16.06 MB)
Trainable params: 4210090 (16.06 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

In [13]:
```
epochs = 5
batch_size = 32
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=batch_size)
```

```
Epoch 1/5
1563/1563 [==============================] - 22s 7ms/step - loss: 1.7150 - accuracy: 0.3781 - val_loss: 1.4045 - val_accurac
y: 0.4967
Epoch 2/5
1563/1563 [==============================] - 11s 7ms/step - loss: 1.3629 - accuracy: 0.5113 - val_loss: 1.2710 - val_accurac
y: 0.6220
Epoch 4/5
1563/1563 [==============================] - 10s 6ms/step - loss: 1.0425 - accuracy: 0.6310 - val_loss: 0.9924 - val_accurac
y: 0.6471
Epoch 5/5
1563/1563 [==============================] - 10s 6ms/step - loss: 0.9334 - accuracy: 0.6717 - val_loss: 1.0220 - val_accurac
y: 0.6380
```

Out[13]: <keras.src.callbacks.History at 0x7b9898366bc0>

In [14]:
```
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 63.80%
```

In [18]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.constraints import max_norm
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.callbacks import LearningRateScheduler

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

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

# Normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
```

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

# Create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu', kernel_constraint=max_norm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu', kernel_constraint=max_norm(3)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu', kernel_constraint=max_norm(3)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

# Compile model
import tensorflow as tf
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers.schedules import ExponentialDecay
from tensorflow.keras.callbacks import LearningRateScheduler

# Define a learning rate schedule using ExponentialDecay
initial_learning_rate = 0.01
decay_steps = 10000
decay_rate = 0.9
```

```
learning_rate_schedule = ExponentialDecay(
    initial_learning_rate, decay_steps, decay_rate, staircase=True
)

# Create the SGD optimizer with the learning rate schedule
sgd = SGD(learning_rate=learning_rate_schedule, momentum=0.9)

# Compile your model
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

# Print the model summary
print(model.summary())

# Fit the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)

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

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 32, 32, 32)        896

 dropout_8 (Dropout)         (None, 32, 32, 32)        0

 conv2d_9 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_4 (MaxPoolin  (None, 16, 16, 32)        0
 g2D)

 conv2d_10 (Conv2D)          (None, 16, 16, 64)        18496

 dropout_9 (Dropout)         (None, 16, 16, 64)        0

 conv2d_11 (Conv2D)          (None, 16, 16, 64)        36928
```

```
max_pooling2d_5 (MaxPoolin  (None, 8, 8, 64)          0
g2D)

conv2d_12 (Conv2D)          (None, 8, 8, 128)         73856

dropout_10 (Dropout)        (None, 8, 8, 128)         0

conv2d_13 (Conv2D)          (None, 8, 8, 128)         147584

max_pooling2d_6 (MaxPoolin  (None, 4, 4, 128)         0
g2D)

flatten_2 (Flatten)         (None, 2048)              0

dropout_11 (Dropout)        (None, 2048)              0

dense_5 (Dense)             (None, 1024)              2098176

dropout_12 (Dropout)        (None, 1024)              0

dense_6 (Dense)             (None, 512)               524800

dropout_13 (Dropout)        (None, 512)               0

dense_7 (Dense)             (None, 10)                5130

=================================================================
Total params: 2915114 (11.12 MB)
Trainable params: 2915114 (11.12 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Epoch 1/5
1563/1563 [==============================] - 17s 10ms/step - loss: 1.8792 - accuracy: 0.3045 - val_loss: 1.5596 - val_accurac
y: 0.4225
Epoch 2/5
1563/1563 [==============================] - 14s 9ms/step - loss: 1.4506 - accuracy: 0.4716 - val_loss: 1.2861 - val_accurac
y: 0.5369
Epoch 3/5
1563/1563 [==============================] - 14s 9ms/step - loss: 1.2423 - accuracy: 0.5513 - val_loss: 1.1531 - val_accurac
y: 0.5921
Epoch 4/5
1563/1563 [==============================] - 14s 9ms/step - loss: 1.0784 - accuracy: 0.6183 - val_loss: 0.9988 - val_accurac
y: 0.6422
Epoch 5/5
1563/1563 [==============================] - 14s 9ms/step - loss: 0.9540 - accuracy: 0.6647 - val_loss: 0.9601 - val_accurac
y: 0.6663
Accuracy: 66.63%
```

In [21]:

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

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

```
1/1 [==============================] - 0s 37ms/step
Predicted labels: [3 8 8 0]
Actual labels:    [3 8 8 0]
```

In [22]:

```
import matplotlib.pyplot as plt

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

# Plot the training and validation accuracy
```

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