# ICP 4

Name : Prashanth Reddy Koppula
700#   : 700761149

**Google Drive Link:**
**https://drive.google.com/file/d/17IWEQkLGwPPwESZJ0jHmsD1-aE8FTimf/view?usp=drive_link**

**Github Link : https://github.com/Nagi-131/700761149_ICP4**

```python
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
```

```python
np.random.seed(7)
```

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

```python
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
```

```python
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
num_classes = y_test.shape[1]
```

```python
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'))
```

```python
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'))
```

```python
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_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 32, 32, 32)        896

 dropout_20 (Dropout)        (None, 32, 32, 32)        0

 conv2d_21 (Conv2D)          (None, 32, 32, 32)        9248

 max_pooling2d_10 (MaxPooli  (None, 16, 16, 32)        0
 ng2D)

 flatten_4 (Flatten)         (None, 8192)              0

 dense_11 (Dense)            (None, 512)               4194816

 dropout_21 (Dropout)        (None, 512)               0

 dense_12 (Dense)            (None, 10)                5130

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

```
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 [==============================] - 13s 8ms/step - loss: 0.8763 - accuracy: 0.6911 - val_loss: 0.8648 - val_accuracy: 0.6949
Epoch 2/5
```

```
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 [==============================] - 13s 8ms/step - loss: 0.8763 - accuracy: 0.6911 - val_loss: 0.8648 - val_accuracy: 0.6949
Epoch 2/5
1563/1563 [==============================] - 13s 8ms/step - loss: 0.7826 - accuracy: 0.7258 - val_loss: 0.7734 - val_accuracy: 0.7328
Epoch 3/5
1563/1563 [==============================] - 13s 8ms/step - loss: 0.7218 - accuracy: 0.7463 - val_loss: 0.7282 - val_accuracy: 0.7503
Epoch 4/5
1563/1563 [==============================] - 13s 8ms/step - loss: 0.6827 - accuracy: 0.7608 - val_loss: 0.7470 - val_accuracy: 0.7441
Epoch 5/5
1563/1563 [==============================] - 13s 8ms/step - loss: 0.6384 - accuracy: 0.7754 - val_loss: 0.6955 - val_accuracy: 0.7604
<keras.src.callbacks.History at 0x7a9889fb7970>
```

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

```
Accuracy: 63.77%
```

```
[ ] 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)))
```

```python
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))
```

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

```
 conv2d_22 (Conv2D)          (None, 32, 32, 32)        896

 dropout_22 (Dropout)        (None, 32, 32, 32)        0

 conv2d_23 (Conv2D)          (None, 32, 32, 32)        9248

 max_pooling2d_11 (MaxPooli  (None, 16, 16, 32)        0
 ng2D)

 conv2d_24 (Conv2D)          (None, 16, 16, 64)        18496

 dropout_23 (Dropout)        (None, 16, 16, 64)        0

 conv2d_25 (Conv2D)          (None, 16, 16, 64)        36928

 max_pooling2d_12 (MaxPooli  (None, 8, 8, 64)          0
 ng2D)

 conv2d_26 (Conv2D)          (None, 8, 8, 128)         73856

 dropout_24 (Dropout)        (None, 8, 8, 128)         0

 conv2d_27 (Conv2D)          (None, 8, 8, 128)         147584

 max_pooling2d_13 (MaxPooli  (None, 4, 4, 128)         0
 ng2D)

 flatten_5 (Flatten)         (None, 2048)              0

 dropout_25 (Dropout)        (None, 2048)              0

 dense_13 (Dense)            (None, 1024)              2098176
```

```
dropout_26 (Dropout)        (None, 1024)             0

dense_14 (Dense)            (None, 512)              524800

dropout_27 (Dropout)        (None, 512)              0

dense_15 (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 [==============================] - 16s 9ms/step - loss: 1.8382 - accuracy: 0.3206 - val_loss: 1.5449 - val_accuracy: 0.4274
Epoch 2/5
1563/1563 [==============================] - 13s 9ms/step - loss: 1.4121 - accuracy: 0.4876 - val_loss: 1.2685 - val_accuracy: 0.5505
Epoch 3/5
1563/1563 [==============================] - 13s 9ms/step - loss: 1.1856 - accuracy: 0.5775 - val_loss: 1.0433 - val_accuracy: 0.6374
Epoch 4/5
1563/1563 [==============================] - 13s 8ms/step - loss: 1.0258 - accuracy: 0.6382 - val_loss: 0.9576 - val_accuracy: 0.6645
Epoch 5/5
1563/1563 [==============================] - 13s 8ms/step - loss: 0.9101 - accuracy: 0.6772 - val_loss: 0.8770 - val_accuracy: 0.6967
Accuracy: 69.67%
```

## New Section

```
[3] # Predict the first 4 images of the test data
    predictions = model.predict(X_test[:4])
    # Convert the predictions to class labels
```


```
# 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)
```

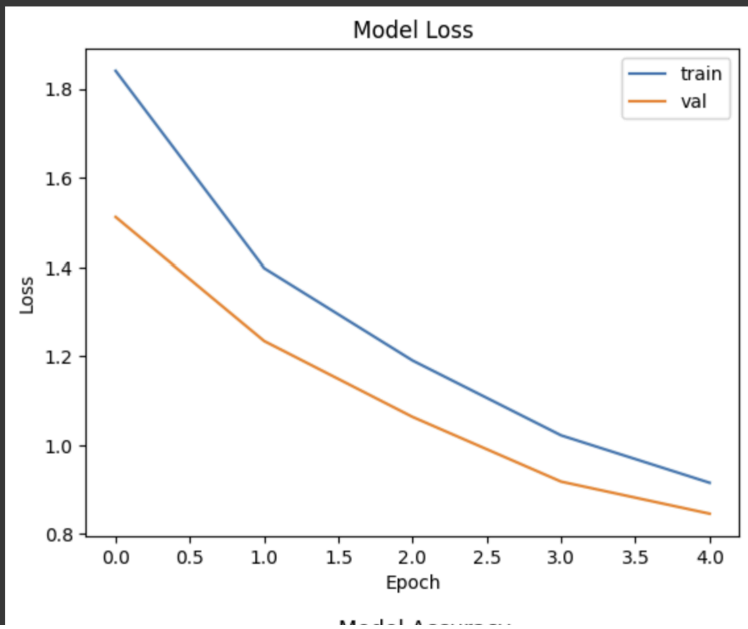
```
1/1 [==============================] - 0s 17ms/step
Predicted labels: [3 8 8 0]
Actual labels:    [3 8 8 0]
```

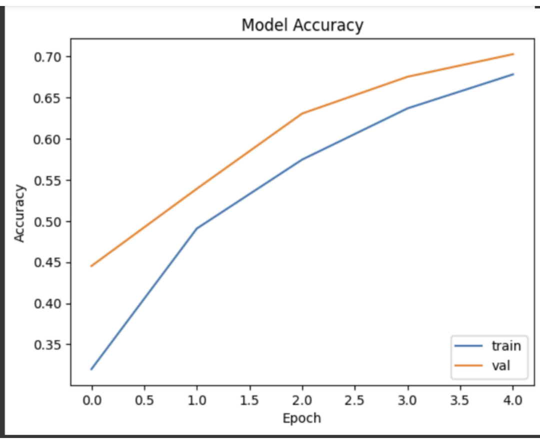
```
[ ] import matplotlib.pyplot as plt

    # 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')
```

```python
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='lower right')
plt.show()
```



```python
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=MaxNorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=MaxNorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=MaxNorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=MaxNorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=MaxNorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu', kernel_constraint=MaxNorm(3)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu', kernel_constraint=MaxNorm(3)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

# Compile model
epochs = 25
lrate = 0.01
lr_schedule = ExponentialDecay(
    initial_learning_rate=lrate,
    decay_steps=epochs * len(X_train) // 32,
    decay_rate=0.1
)
sgd = SGD(learning_rate=lr_schedule, momentum=0.9, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())

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

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

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

# Predict the first 4 images of the test data
predictions = model.predict(X_test[:4])
predicted_classes = np.argmax(predictions, axis=1)
actual_classes = np.argmax(y_test[:4], axis=1)


# Print the predictions and actual labels
print("Predicted classes: ", predicted_classes)
print("Actual classes: ", actual_classes)

# Plot the first 4 test images, predicted labels, and actual labels
fig, axes = plt.subplots(1, 4, figsize=(15, 3))
for i in range(4):
    axes[i].imshow(X_test[i])
    axes[i].set_title(f"Pred: {predicted_classes[i]}, Actual: {actual_classes[i]}")
    axes[i].axis('off')
# Visualize loss and accuracy
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

```
# Visualize loss and accuracy
plt.figure(figsize=(12, 4))

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

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

plt.show()
```

```
Model: "sequential_2"

 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 32, 32, 32)        896

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

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

 max_pooling2d_6 (MaxPoolin  (None, 16, 16, 32)        0
 g2D)

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

```
plt.legend()

plt.show()
```

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

 max_pooling2d_7 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

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

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

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

 max_pooling2d_8 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 flatten_2 (Flatten)         (None, 2048)              0

 dropout_15 (Dropout)        (None, 2048)              0

 dense_6 (Dense)             (None, 1024)              2098176

 dropout_16 (Dropout)        (None, 1024)              0

 dense_7 (Dense)             (None, 512)               524800

 dropout_17 (Dropout)        (None, 512)               0

 dense_8 (Dense)             (None, 10)                5130

=================================================================
Total params: 2915114 (11.12 MB)
Trainable params: 2915114 (11.12 MB)
```

```python
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

```
Total params: 2915114 (11.12 MB)
Trainable params: 2915114 (11.12 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Epoch 1/25
1563/1563 [==============================] - 16s 9ms/step - loss: 1.8386 - accuracy: 0.3188 - val_loss: 1.4968 - val_accuracy: 0.4601
Epoch 2/25
1563/1563 [==============================] - 14s 9ms/step - loss: 1.4173 - accuracy: 0.4825 - val_loss: 1.3006 - val_accuracy: 0.5102
Epoch 3/25
1563/1563 [==============================] - 15s 10ms/step - loss: 1.2150 - accuracy: 0.5634 - val_loss: 1.0621 - val_accuracy: 0.6180
Epoch 4/25
1563/1563 [==============================] - 13s 9ms/step - loss: 1.0480 - accuracy: 0.6256 - val_loss: 0.9433 - val_accuracy: 0.6711
Epoch 5/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.9200 - accuracy: 0.6745 - val_loss: 0.8729 - val_accuracy: 0.6929
Epoch 6/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.8212 - accuracy: 0.7126 - val_loss: 0.7843 - val_accuracy: 0.7227
Epoch 7/25
1563/1563 [==============================] - 13s 9ms/step - loss: 0.7419 - accuracy: 0.7385 - val_loss: 0.7602 - val_accuracy: 0.7375
Epoch 8/25
1563/1563 [==============================] - 14s 9ms/step - loss: 0.6674 - accuracy: 0.7635 - val_loss: 0.7044 - val_accuracy: 0.7568
Epoch 9/25
1563/1563 [==============================] - 14s 9ms/step - loss: 0.6045 - accuracy: 0.7858 - val_loss: 0.6734 - val_accuracy: 0.7674
Epoch 10/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.5523 - accuracy: 0.8058 - val_loss: 0.6534 - val_accuracy: 0.7745
Epoch 11/25
1563/1563 [==============================] - 16s 10ms/step - loss: 0.4989 - accuracy: 0.8251 - val_loss: 0.6362 - val_accuracy: 0.7830
Epoch 12/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.4566 - accuracy: 0.8369 - val_loss: 0.6486 - val_accuracy: 0.7768
Epoch 13/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.4161 - accuracy: 0.8517 - val_loss: 0.6417 - val_accuracy: 0.7788
```

```
Epoch 14/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.3816 - accuracy: 0.8640 - val_loss: 0.6313 - val_accuracy: 0.7919
Epoch 15/25
1563/1563 [==============================] - 13s 9ms/step - loss: 0.3470 - accuracy: 0.8759 - val_loss: 0.6221 - val_accuracy: 0.7917
Epoch 16/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.3164 - accuracy: 0.8852 - val_loss: 0.6307 - val_accuracy: 0.7973
Epoch 17/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.2877 - accuracy: 0.8964 - val_loss: 0.6316 - val_accuracy: 0.8027
Epoch 18/25
1563/1563 [==============================] - 13s 9ms/step - loss: 0.2660 - accuracy: 0.9056 - val_loss: 0.6335 - val_accuracy: 0.8043
Epoch 19/25
1563/1563 [==============================] - 15s 9ms/step - loss: 0.2416 - accuracy: 0.9136 - val_loss: 0.6440 - val_accuracy: 0.8057
Epoch 20/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.2269 - accuracy: 0.9195 - val_loss: 0.6688 - val_accuracy: 0.8027
Epoch 21/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.2068 - accuracy: 0.9249 - val_loss: 0.6542 - val_accuracy: 0.8056
Epoch 22/25
1563/1563 [==============================] - 13s 9ms/step - loss: 0.1894 - accuracy: 0.9326 - val_loss: 0.6842 - val_accuracy: 0.8070
Epoch 23/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.1824 - accuracy: 0.9340 - val_loss: 0.6747 - val_accuracy: 0.8102
Epoch 24/25
1563/1563 [==============================] - 13s 9ms/step - loss: 0.1667 - accuracy: 0.9399 - val_loss: 0.6862 - val_accuracy: 0.8068
Epoch 25/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.1543 - accuracy: 0.9449 - val_loss: 0.6939 - val_accuracy: 0.8092
Accuracy: 80.92%
1/1 [==============================] - 0s 94ms/step
Predicted classes:  [5 8 8 0]
Actual classes:  [3 8 8 0]
```

```
Epoch 25/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.1543 - accuracy: 0.9449 - val_loss: 0.6939 - val_accuracy: 0.8092
Accuracy: 80.92%
1/1 [==============================] - 0s 94ms/step
Predicted classes:  [5 8 8 0]
Actual classes:  [3 8 8 0]
```