Project I | Deep Learning: Image Classification with CNN

(Week 6)

Student: Caroline Araujo

The Dataset

THE DATASET: OVERVIEW

< Back to Alex Krizhevsky's home page

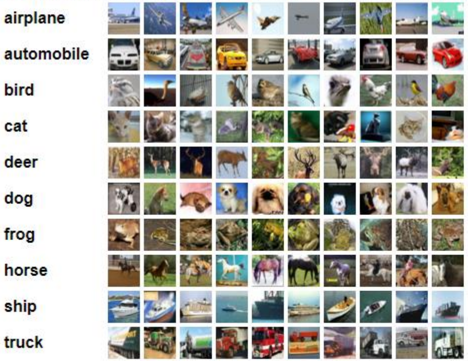
The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

The CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:



The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

- ➔ Extracted from the website https://www.cs.toronto.edu/~kriz/cifar.html
- ➔ Created by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton
- ➔ University of Toronto's Deep Learning Lab
- ➔ It was introduced in 2009 as part of a project to facilitate research in deep learning and computer vision

THE DATASET: CONTENT

< Back to Alex Krizhevsky's home page

The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

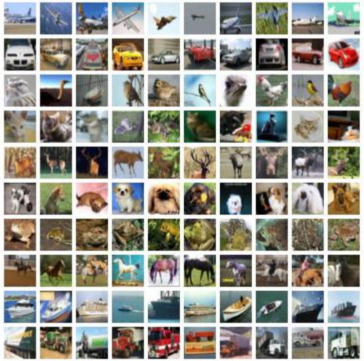
The CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck



The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

➔ 60,000 images

- Color images
- 32x32 pixels
- Distributed across 10 categories
- With 6,000 images per category

➔ 50,000 in the training
➔ 10,000 in the test

Data Preprocessing

IMPORTING LIBRARIES, LOADING AND DISPLAYING THE DATA

Import libraries

```python
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import image
from tensorflow.keras.datasets import cifar10
from skimage import color, data, exposure, filters, io, morphology
```

[2] ✓ 6.7s Python

Load the data

```python
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

[3] ✓ 9.8s Python

Display the data

```python
# Display information about the dataset
print(f'Training data shape: {x_train.shape}')
print(f'Training labels shape: {y_train.shape}')
print(f'Test data shape: {x_test.shape}')
print(f'Test labels shape: {y_test.shape}')
```

[4] ✓ 0.2s Python

```
Training data shape: (50000, 32, 32, 3)
Training labels shape: (50000, 1)
Test data shape: (10000, 32, 32, 3)
Test labels shape: (10000, 1)
```

NORMALIZATION AND DATA AUGMENTATION

Normalization

```python
# Convert to float32 type and normalize (dividing by 255)
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

[6] ✓ 13.5s Python

Data Augmentation

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Creating a data generator for data augmentation
datagen = ImageDataGenerator(
    rotation_range=15,        # Random rotation up to 15 degrees
    width_shift_range=0.1,    # Horizontal displacement up to 10%
    height_shift_range=0.1,   # Vertical displacement up to 10%
    horizontal_flip=True      # Horizontal turning
)

# Adjusting the generator to the training data
datagen.fit(x_train)
```

[8] ✓ 0.5s Python

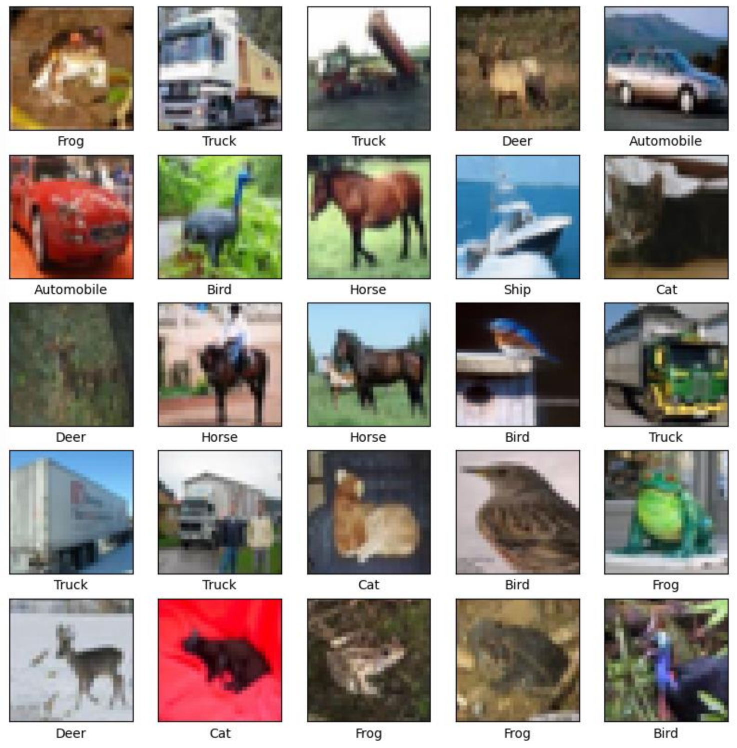
VISUALIZATIONS OF SOME IMAGES AND LABELS

View part of the data

```python
# View some images of the training set
classes = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer',
           'Dog', 'Frog', 'Horse', 'Ship', 'Truck' ]

plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i])
    # Convert the label to a number and use it to get the class name
    plt.xlabel(classes[y_train[i][0]])
plt.show()
```

[5] ✓ 4.5s Python



Model Training

Model Training

```python
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Define callbacks
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=10,         # Wait 10 epochs before stopping
    restore_best_weights=True  # Restore the best weights at the end
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',  # Monitor validation loss
    factor=0.2,          # Reduce the learning rate by a factor of 0.2
    patience=5,          # Wait 5 seasons without improvement before cutting back
    min_lr=1e-5          # Lower limit for learning rate
)

# Training the model
history = model.fit(
    x_train, y_train,              # Training data
    epochs=5,                      # Maximum number of epochs
    batch_size=32,                 # Lot size
    validation_data=(x_test, y_test), # Validation data
    callbacks=[early_stopping, reduce_lr]  # Add callbacks
)
```

[19] ✓ 67m 56.5s Python

```
Epoch 1/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 725s 456ms/step - accuracy: 0.2365 - loss: 2.1158 - val_accuracy: 0.4904 - val_loss: 1.4845 - learning_rate: 1.0000e-04
Epoch 2/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 846s 541ms/step - accuracy: 0.4483 - loss: 1.5715 - val_accuracy: 0.5254 - val_loss: 1.3541 - learning_rate: 1.0000e-04
Epoch 3/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 1001s 640ms/step - accuracy: 0.4965 - loss: 1.4370 - val_accuracy: 0.5481 - val_loss: 1.2880 - learning_rate: 1.0000e-04
Epoch 4/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 970s 620ms/step - accuracy: 0.5264 - loss: 1.3540 - val_accuracy: 0.5589 - val_loss: 1.2497 - learning_rate: 1.0000e-04
Epoch 5/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 524s 335ms/step - accuracy: 0.5441 - loss: 1.3153 - val_accuracy: 0.5728 - val_loss: 1.2181 - learning_rate: 1.0000e-04
```

Training the model with 5 epochs

Training the model with 20 epochs

```python
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Define callbacks
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=10,         # Wait 10 epochs before stopping
    restore_best_weights=True  # Restore the best weights at the end
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',  # Monitor validation loss
    factor=0.2,          # Reduce the learning rate by a factor of 0.2
    patience=5,          # Wait 5 seasons without improvement before cutting back
    min_lr=1e-5          # Lower limit for learning rate
)

# Training the model
history = model.fit(
    x_train, y_train,                        # Training data
    epochs=20,                                # Maximum number of epochs
    batch_size=32,                           # Lot size
    validation_data=(x_test, y_test), # Validation data
    callbacks=[early_stopping, reduce_lr]  # Add callbacks
)
```

[13] ✓ 54m 8.6s

```
Epoch 1/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 133s 85ms/step - accuracy: 0.6354 - loss: 1.0396 - val_accuracy: 0.6768 - val_loss: 0.9376 - learning_rate: 0.0010
Epoch 2/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 138s 88ms/step - accuracy: 0.6485 - loss: 1.0098 - val_accuracy: 0.6631 - val_loss: 0.9740 - learning_rate: 0.0010
Epoch 3/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 141s 90ms/step - accuracy: 0.6595 - loss: 0.9846 - val_accuracy: 0.6916 - val_loss: 0.8936 - learning_rate: 0.0010
Epoch 4/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 147s 94ms/step - accuracy: 0.6642 - loss: 0.9583 - val_accuracy: 0.6951 - val_loss: 0.8808 - learning_rate: 0.0010
Epoch 5/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 147s 94ms/step - accuracy: 0.6744 - loss: 0.9472 - val_accuracy: 0.6905 - val_loss: 0.9061 - learning_rate: 0.0010
Epoch 6/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 196s 90ms/step - accuracy: 0.6799 - loss: 0.9331 - val_accuracy: 0.6967 - val_loss: 0.8795 - learning_rate: 0.0010
Epoch 7/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 139s 89ms/step - accuracy: 0.6796 - loss: 0.9182 - val_accuracy: 0.6966 - val_loss: 0.8921 - learning_rate: 0.0010
Epoch 8/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 140s 90ms/step - accuracy: 0.6899 - loss: 0.8993 - val_accuracy: 0.7088 - val_loss: 0.8477 - learning_rate: 0.0010
Epoch 9/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 149s 95ms/step - accuracy: 0.6951 - loss: 0.8849 - val_accuracy: 0.7154 - val_loss: 0.8238 - learning_rate: 0.0010
Epoch 10/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 153s 98ms/step - accuracy: 0.6955 - loss: 0.8884 - val_accuracy: 0.7039 - val_loss: 0.8861 - learning_rate: 0.0010
Epoch 11/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 203s 98ms/step - accuracy: 0.6949 - loss: 0.8836 - val_accuracy: 0.7210 - val_loss: 0.8218 - learning_rate: 0.0010
Epoch 12/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 154s 98ms/step - accuracy: 0.6992 - loss: 0.8769 - val_accuracy: 0.7138 - val_loss: 0.8368 - learning_rate: 0.0010
Epoch 13/20
...
Epoch 19/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 185s 119ms/step - accuracy: 0.7155 - loss: 0.8318 - val_accuracy: 0.7256 - val_loss: 0.8144 - learning_rate: 0.0010
Epoch 20/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 155s 99ms/step - accuracy: 0.7158 - loss: 0.8292 - val_accuracy: 0.7311 - val_loss: 0.7905 - learning_rate: 0.0010
```

Display the results

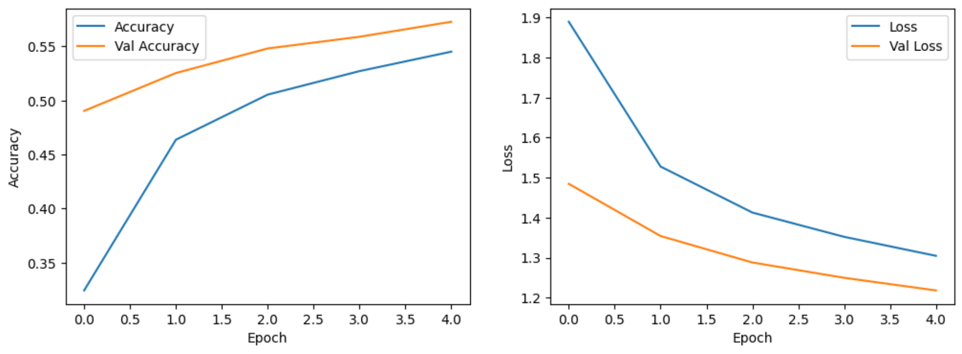
```python
import matplotlib.pyplot as plt

# Accuracy graph
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss graph
plt.subplot(1, 2, 2)
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.show()
```

[20] ✓ 0.5s Python

Displaying of results with 5 epochs



Display the results

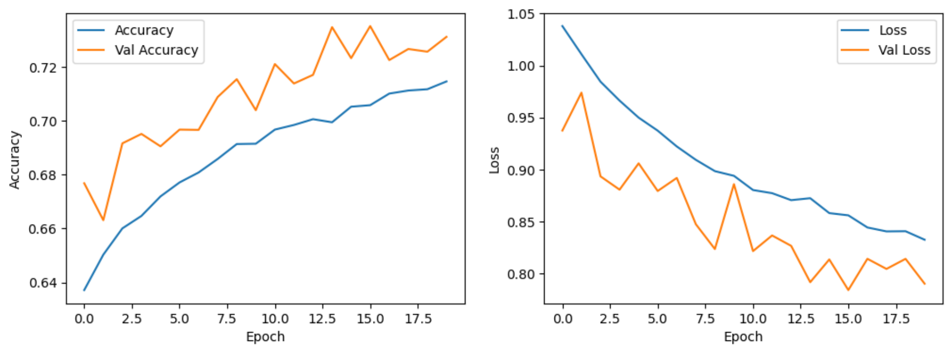
```python
import matplotlib.pyplot as plt

# Accuracy graph
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss graph
plt.subplot(1, 2, 2)
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.show()
```

[20] ✓ 0.5s Python

Displaying of results with 20 epochs



Model Evaluation

Test accuracy with 5 epochs

```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Test Accuracy: {test_accuracy:.4f}")
```

[21] ✓ 1m 57.1s Python

```
313/313 - 116s - 371ms/step - accuracy: 0.5728 - loss: 1.2181
Test Accuracy: 0.5728
```

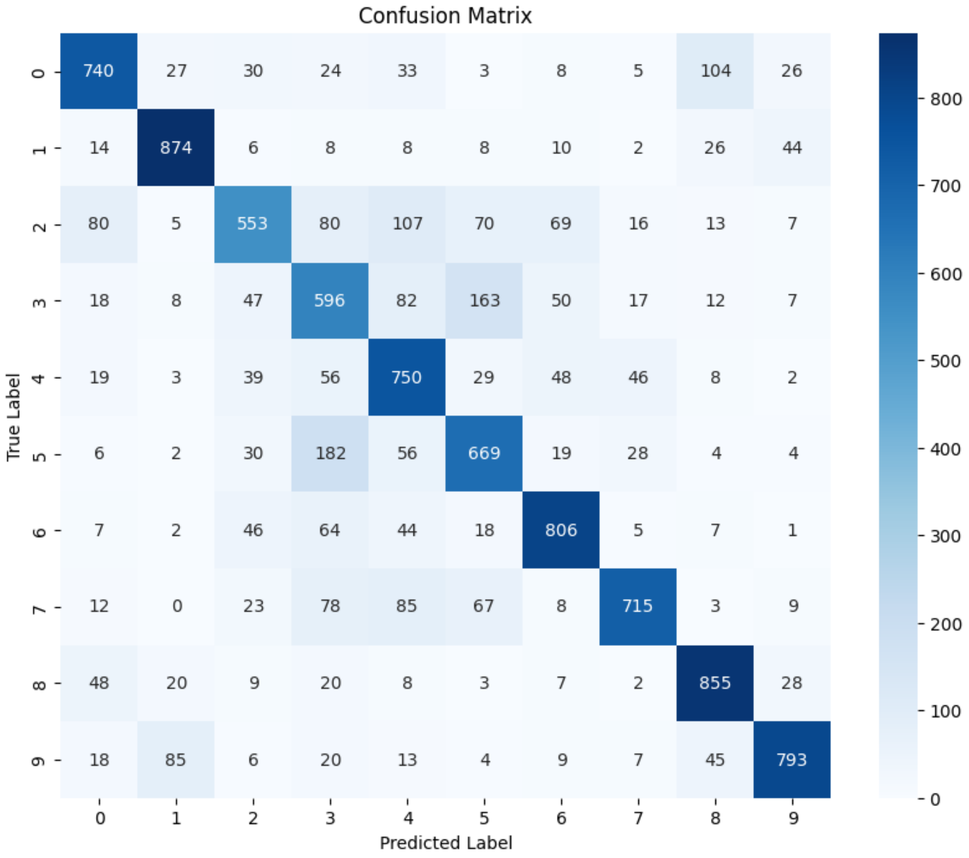
Test accuracy with 20 epochs

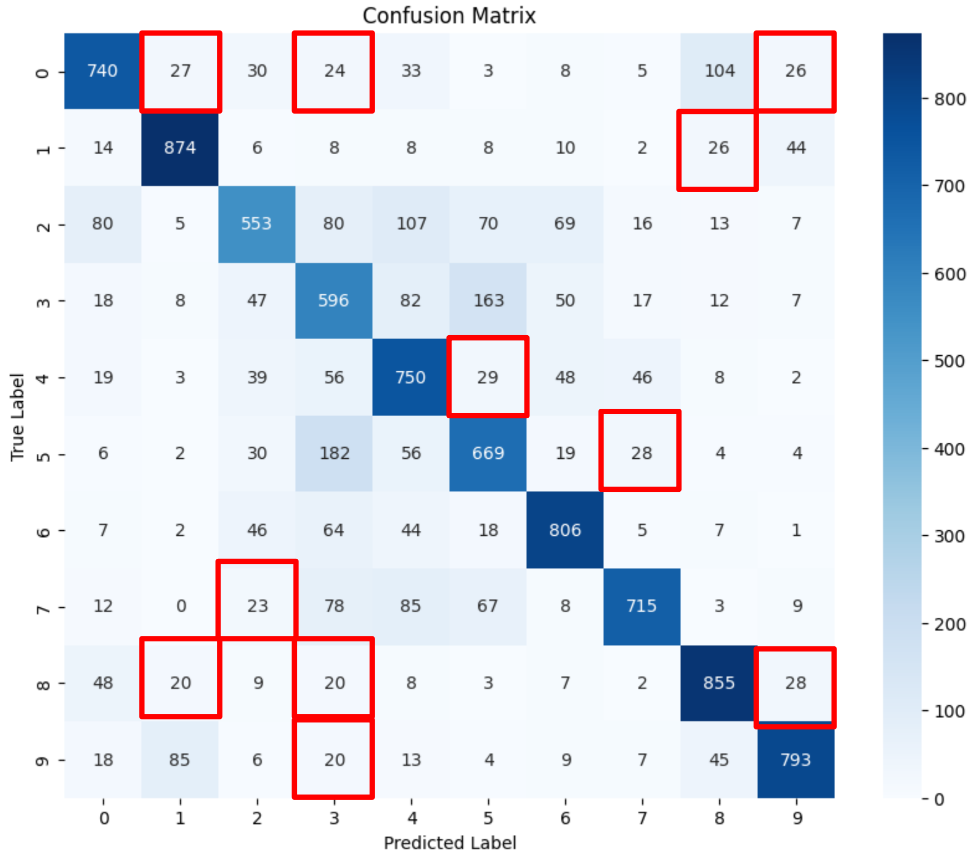
```python
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Test Accuracy: {test_accuracy:.4f}")
```

[15] ✓ 7.7s Python

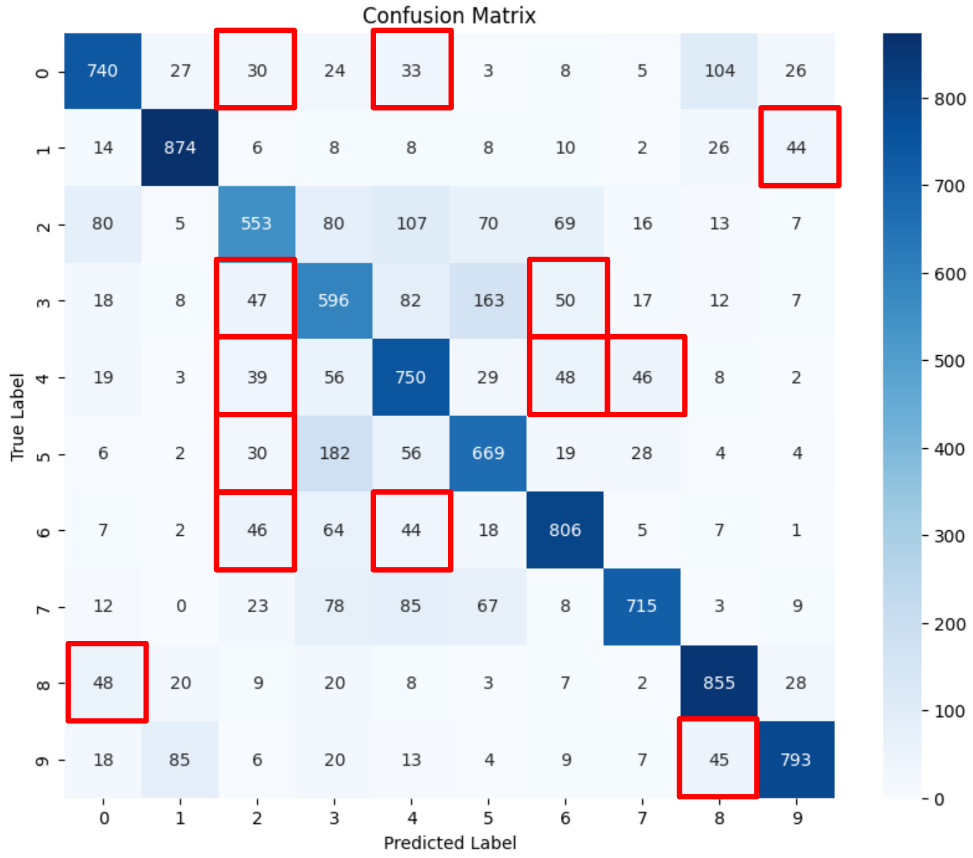
```
313/313 - 6s - 20ms/step - accuracy: 0.7351 - loss: 0.7844
Test Accuracy: 0.7351
```

The Confusion Matrix

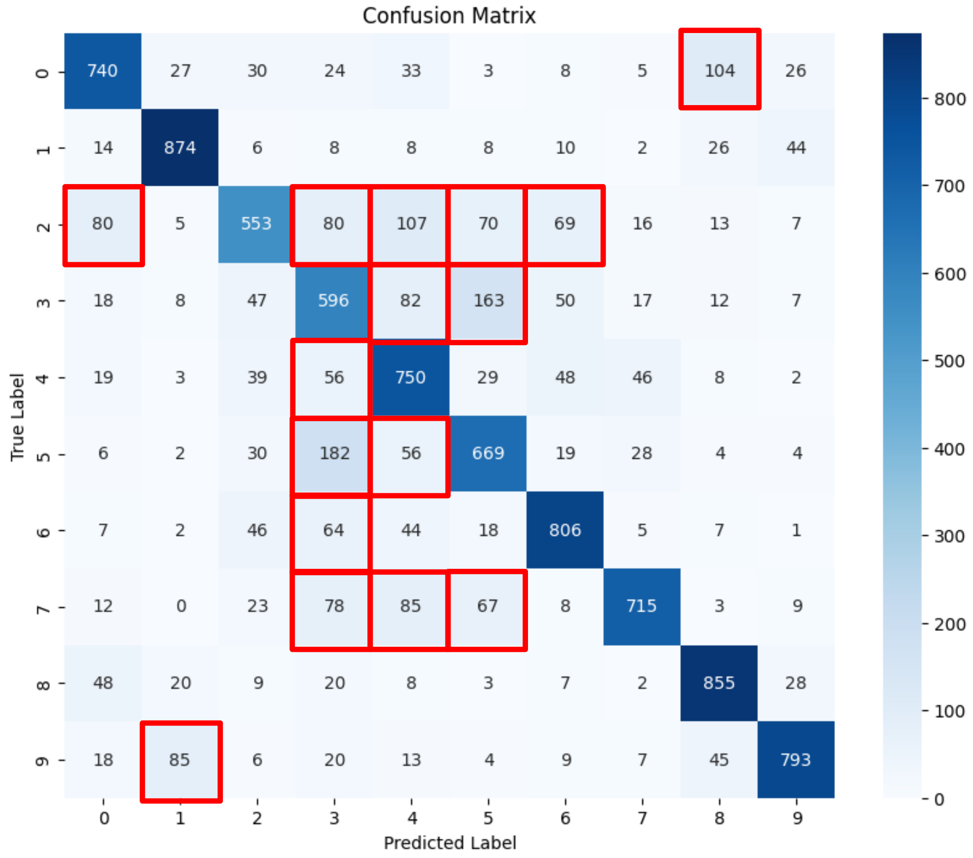




Up to 20-30 errors per class: acceptable



Between 30-50 errors per class: can be improved



Above 50 errors per class: doesn’t work as it should

Model Performance Metrics

Metrics Calculation (Precision, Recall and F1-Score)

```python
from sklearn.metrics import classification_report

# Obtain predictions on the test set
y_pred = model.predict(x_test)
y_pred_classes = y_pred.argmax(axis=1)

# Generate a classification report
print(classification_report(y_test, y_pred_classes))
```

[18] ✓ 1m 55.7s Python

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 114s 362ms/step
                  precision    recall  f1-score   support

 ['Airplane',  0       0.68      0.72      0.70      1000
 'Automobile', 1       0.69      0.70      0.70      1000
       'Bird', 2       0.57      0.48      0.52      1000
        'Cat', 3       0.47      0.50      0.49      1000
       'Deer', 4       0.57      0.57      0.57      1000
        'Dog', 5       0.63      0.53      0.57      1000
       'Frog', 6       0.65      0.73      0.69      1000
      'Horse', 7       0.69      0.67      0.68      1000
       'Ship', 8       0.73      0.73      0.73      1000
      'Truck' ] 9      0.64      0.69      0.66      1000

        accuracy                           0.63     10000
       macro avg       0.63      0.63      0.63     10000
    weighted avg       0.63      0.63      0.63     10000
```

Precision per class

Transfer Learning

```python
# Training the model with Transfer Learning
history = model.fit(
    x_train, y_train,
    epochs=20,
    batch_size=32,
    validation_data=(x_test, y_test),
    callbacks=[early_stopping, reduce_lr]
)
```

[14] ✓ 182m 52.3s Python

```
Epoch 1/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 478s 304ms/step - accuracy: 0.1872 - loss: 13.5271 - val_accuracy: 0.2114 - val_loss: 2.1875 - learning_rate: 1.0000e-04
Epoch 2/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 446s 285ms/step - accuracy: 0.2132 - loss: 2.4738 - val_accuracy: 0.3177 - val_loss: 2.0388 - learning_rate: 1.0000e-04
Epoch 3/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 504s 287ms/step - accuracy: 0.2727 - loss: 2.0863 - val_accuracy: 0.4113 - val_loss: 1.7796 - learning_rate: 1.0000e-04
Epoch 4/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 467s 299ms/step - accuracy: 0.3390 - loss: 1.8680 - val_accuracy: 0.4708 - val_loss: 1.6096 - learning_rate: 1.0000e-04
Epoch 5/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 447s 286ms/step - accuracy: 0.3827 - loss: 1.7453 - val_accuracy: 0.5083 - val_loss: 1.4940 - learning_rate: 1.0000e-04
Epoch 6/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 472s 302ms/step - accuracy: 0.4212 - loss: 1.6380 - val_accuracy: 0.5290 - val_loss: 1.4268 - learning_rate: 1.0000e-04
Epoch 7/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 480s 307ms/step - accuracy: 0.4554 - loss: 1.5599 - val_accuracy: 0.5499 - val_loss: 1.3700 - learning_rate: 1.0000e-04
Epoch 8/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 458s 293ms/step - accuracy: 0.4827 - loss: 1.4785 - val_accuracy: 0.5633 - val_loss: 1.3192 - learning_rate: 1.0000e-04
Epoch 9/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 456s 292ms/step - accuracy: 0.4988 - loss: 1.4336 - val_accuracy: 0.5769 - val_loss: 1.2906 - learning_rate: 1.0000e-04
Epoch 10/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 452s 289ms/step - accuracy: 0.5215 - loss: 1.3873 - val_accuracy: 0.5908 - val_loss: 1.2560 - learning_rate: 1.0000e-04
Epoch 11/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 507s 292ms/step - accuracy: 0.5415 - loss: 1.3309 - val_accuracy: 0.5950 - val_loss: 1.2286 - learning_rate: 1.0000e-04
Epoch 12/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 452s 289ms/step - accuracy: 0.5478 - loss: 1.3076 - val_accuracy: 0.6020 - val_loss: 1.2051 - learning_rate: 1.0000e-04
Epoch 13/20
...
Epoch 19/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 559s 358ms/step - accuracy: 0.6140 - loss: 1.1181 - val_accuracy: 0.6290 - val_loss: 1.1214 - learning_rate: 1.0000e-04
Epoch 20/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 703s 450ms/step - accuracy: 0.6167 - loss: 1.0921 - val_accuracy: 0.6325 - val_loss: 1.1089 - learning_rate: 1.0000e-04
```

... and the test accuracy was:

```
# Evaluation of the transferred model
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Transfer Learning Model Accuracy: {test_accuracy:.4f}")
```

[16] ✓ 1m 42.4s Python

```
313/313 - 102s - 327ms/step - accuracy: 0.6325 - loss: 1.1089
Transfer Learning Model Accuracy: 0.6325
```

```python
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Test Accuracy: {test_accuracy:.4f}")
```

[21] ✓ 1m 57.1s Python

```
313/313 - 116s - 371ms/step - accuracy: 0.5728 - loss: 1.2181
Test Accuracy: 0.5728
```

1st model with 5 epochs

```python
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Test Accuracy: {test_accuracy:.4f}")
```

[15] ✓ 7.7s Python

```
313/313 - 6s - 20ms/step - accuracy: 0.7351 - loss: 0.7844
Test Accuracy: 0.7351
```

1st model with 20 epochs

Transfer learning model

```python
# Evaluation of the transferred model
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Transfer Learning Model Accuracy: {test_accuracy:.4f}")
```

[16] ✓ 1m 42.4s Python

```
313/313 - 102s - 327ms/step - accuracy: 0.6325 - loss: 1.1089
Transfer Learning Model Accuracy: 0.6325
```

Business-oriented approach

Error Impact Per Class
Impact
Moderate
Low
High
Number of Errors
0
50
100
150
200
250
300
350
400
Airplane
Automobile
Bird
Cat
Deer
Dog
Frog
Horse
Ship
Truck
Class

➔ **Low-error classes (e.g., Class 1, 8, 9):** The model performs well here, ensuring accurate product categorization.

➔ **Moderate errors (e.g., Class 3, 5):** Errors in these categories may lead to incorrect recommendations, impacting customer satisfaction.

➔ **High-error classes (e.g., Class 2, 9):** Frequent misclassification of high-value products can result in reduced sales or higher return rates.

IRON HACK

Thank you for your attention!