AlexNet using CIFAR-10

April 3, 2025

[3]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, BatchNormalization
from tensorflow.keras.datasets import cifar10
from tensorflow.keras import optimizers
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
```

[5]:
```
#Loading the data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

[7]:
```
from tensorflow.keras.utils import to_categorical

y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

[9]:
```
#Normalising the data to get the vales in the range [0,1]
x_train, x_test = x_train.astype("float32") / 255.0, x_test.astype("float32") /␣
 ↪255.0
```

[11]:
```
train_datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.1,
)
```

[13]:
```
test_datagen = ImageDataGenerator()
```

[15]:
```
#Generating images for training and testing in the model
train_generator = train_datagen.flow(x_train, y_train, batch_size=128)
test_generator = test_datagen.flow(x_test, y_test, batch_size=128)
```

[17]:
```
model = Sequential([
    Conv2D(96, (3, 3), strides=(1, 1), activation='relu', padding='same',␣
 ↪input_shape=(32, 32, 3)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
    Conv2D(256, (3, 3), padding='same', activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
    Conv2D(384, (3, 3), padding='same', activation='relu'),
    BatchNormalization(),
    Conv2D(384, (3, 3), padding='same', activation='relu'),
    BatchNormalization(),
    Conv2D(256, (3, 3), padding='same', activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.3),
    Dense(256, activation='relu'),
    Dropout(0.3),
    Dense(10, activation='softmax')
])

optimizer=tf.keras.optimizers.AdamW(learning_rate=1e-3, weight_decay=1e-4,␣
 ↪clipvalue=1.0)
model.compile(optimizer=optimizer,loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
/opt/anaconda3/envs/tf-metal/lib/python3.9/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2025-04-03 08:55:48.364464: I metal_plugin/src/device/metal_device.cc:1154]
Metal device set to: Apple M1
2025-04-03 08:55:48.364526: I metal_plugin/src/device/metal_device.cc:296]
systemMemory: 8.00 GB
2025-04-03 08:55:48.364575: I metal_plugin/src/device/metal_device.cc:313]
maxCacheSize: 2.67 GB
2025-04-03 08:55:48.364878: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305]
Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel
may not have been built with NUMA support.
2025-04-03 08:55:48.364900: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0
MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id:
```

<undefined>)

[19]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 96)	2,688
batch_normalization (BatchNormalization)	(None, 32, 32, 96)	384
max_pooling2d (MaxPooling2D)	(None, 15, 15, 96)	0
conv2d_1 (Conv2D)	(None, 15, 15, 256)	221,440
batch_normalization_1 (BatchNormalization)	(None, 15, 15, 256)	1,024
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 256)	0
conv2d_2 (Conv2D)	(None, 7, 7, 384)	885,120
batch_normalization_2 (BatchNormalization)	(None, 7, 7, 384)	1,536
conv2d_3 (Conv2D)	(None, 7, 7, 384)	1,327,488
batch_normalization_3 (BatchNormalization)	(None, 7, 7, 384)	1,536
conv2d_4 (Conv2D)	(None, 7, 7, 256)	884,992
batch_normalization_4 (BatchNormalization)	(None, 7, 7, 256)	1,024
max_pooling2d_2 (MaxPooling2D)	(None, 3, 3, 256)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 512)	1,180,160
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328

```
 dropout_1 (Dropout)              (None, 256)                    0

 dense_2 (Dense)                  (None, 10)                 2,570

Total params: 4,641,290 (17.71 MB)

Trainable params: 4,638,538 (17.69 MB)

Non-trainable params: 2,752 (10.75 KB)
```

[21]:
```
history = model.fit(train_generator, epochs=10, validation_data=test_generator)
```

```
Epoch 1/10
/opt/anaconda3/envs/tf-metal/lib/python3.9/site-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
2025-04-03 08:56:07.571112: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:117]
Plugin optimizer for device_type GPU is enabled.
391/391            89s 221ms/step -
accuracy: 0.1971 - loss: 17.2317 - val_accuracy: 0.2390 - val_loss: 15.3230
Epoch 2/10
391/391            86s 220ms/step -
accuracy: 0.2729 - loss: 17.1955 - val_accuracy: 0.3836 - val_loss: 9.4347
Epoch 3/10
391/391            86s 220ms/step -
accuracy: 0.3409 - loss: 8.0069 - val_accuracy: 0.3791 - val_loss: 5.2993
Epoch 4/10
391/391            87s 223ms/step -
accuracy: 0.3934 - loss: 4.0883 - val_accuracy: 0.4176 - val_loss: 3.5327
Epoch 5/10
391/391            88s 225ms/step -
accuracy: 0.4287 - loss: 3.3027 - val_accuracy: 0.5277 - val_loss: 2.4025
Epoch 6/10
391/391            89s 228ms/step -
accuracy: 0.4528 - loss: 3.3160 - val_accuracy: 0.4090 - val_loss: 3.5003
Epoch 7/10
391/391            90s 230ms/step -
accuracy: 0.4615 - loss: 3.2472 - val_accuracy: 0.3516 - val_loss: 5.5601
Epoch 8/10
```

```
391/391             90s 230ms/step -
accuracy: 0.4846 - loss: 3.6987 - val_accuracy: 0.4589 - val_loss: 5.2175
Epoch 9/10
391/391             96s 247ms/step -
accuracy: 0.4819 - loss: 4.6660 - val_accuracy: 0.4294 - val_loss: 4.6699
Epoch 10/10
391/391             103s 263ms/step -
accuracy: 0.5187 - loss: 4.8957 - val_accuracy: 0.3013 - val_loss: 7.9245
```

[23]:
```
y_test_labels=np.argmax(y_test, axis=1)
y_pred = np.argmax(model.predict(x_test), axis=1)
print("Classification Report:")
print(classification_report(y_test_labels, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test_labels, y_pred))
```

```
313/313             5s 15ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.21      0.97      0.34      1000
           1       0.95      0.35      0.52      1000
           2       0.15      0.51      0.23      1000
           3       0.00      0.00      0.00      1000
           4       0.45      0.01      0.03      1000
           5       0.81      0.05      0.10      1000
           6       0.97      0.16      0.27      1000
           7       0.91      0.25      0.40      1000
           8       0.62      0.26      0.37      1000
           9       0.83      0.45      0.58      1000

    accuracy                           0.30     10000
   macro avg       0.59      0.30      0.28     10000
weighted avg       0.59      0.30      0.28     10000

Confusion Matrix:
[[967   0  27   0   0   0   0   0   4   2]
 [458 355  12   0   0   1   0   1  90  83]
 [487   0 510   0   1   0   0   1   1   0]
 [309   0 666   0   2   8   2   5   4   4]
 [359   0 619   0  13   0   0   8   1   0]
 [259   0 677   0   0  51   3   6   2   2]
 [178   0 649   0   3   0 159   1  10   0]
 [488   0 243   0  10   2   0 253   1   3]
 [675   1  65   0   0   0   0   0 259   0]
 [457  18  33   0   0   1   0   2  43 446]]

/opt/anaconda3/envs/tf-metal/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
```

```
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/anaconda3/envs/tf-metal/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/anaconda3/envs/tf-metal/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

[25]:
```
precision = precision_score(y_test_labels, y_pred, average='weighted')
recall = recall_score(y_test_labels, y_pred, average='weighted')
f1 = f1_score(y_test_labels, y_pred, average='weighted')

print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
```

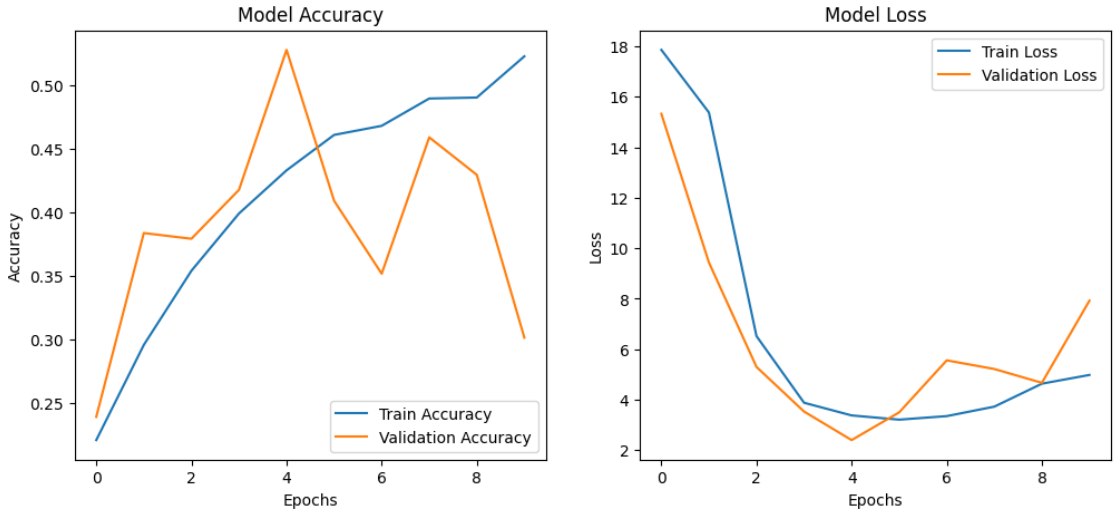
```
Precision: 0.5894
Recall: 0.3013
F1-score: 0.2822

/opt/anaconda3/envs/tf-metal/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

[27]:
```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Model Loss')
```

```
plt.show()
```


Model Accuracy
Accuracy
Epochs
Train Accuracy
Validation Accuracy
Model Loss
Loss
Epochs
Train Loss
Validation Loss