Untitled

November 6, 2024

[45]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import numpy as np
import random
```

[47]:

```
# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

# Normalize pixel values and add a bit of random noise to make images slightly␣
 ↪unique each time they're loaded
x_train, x_test = x_train / 255.0 + np.random.normal(0, 0.01, x_train.shape),␣
 ↪x_test / 255.0 + np.random.normal(0, 0.01, x_test.shape)

# One-hot encode labels
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

[48]:

```
# Define the model architecture
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32,␣
 ↪3)),
    BatchNormalization(),
    Conv2D(32, (3, 3), activation='relu', padding='same'),
    MaxPooling2D((2, 2), padding='same'),  # Use 'same' padding here
    Dropout(0.3),

    Conv2D(64, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    MaxPooling2D((2, 2), padding='same'),  # Use 'same' padding here
    Dropout(0.4),

    Conv2D(128, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    # You can optionally remove the second Conv2D and MaxPooling layers here
    MaxPooling2D((2, 2), padding='same'),  # Final MaxPooling with 'same'␣
 ↪padding
    Dropout(0.5),

    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

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

model.summary()
```

```
Model: "sequential_4"


 Layer (type)                     Output Shape                  Param #

 conv2d_23 (Conv2D)               (None, 32, 32, 32)                896

 batch_normalization_12           (None, 32, 32, 32)                128
 (BatchNormalization)

 conv2d_24 (Conv2D)               (None, 32, 32, 32)              9,248

 max_pooling2d_12 (MaxPooling2D)  (None, 16, 16, 32)                  0

 dropout_16 (Dropout)             (None, 16, 16, 32)                  0

 conv2d_25 (Conv2D)               (None, 16, 16, 64)             18,496

 batch_normalization_13           (None, 16, 16, 64)                256
 (BatchNormalization)

 conv2d_26 (Conv2D)               (None, 16, 16, 64)             36,928

 max_pooling2d_13 (MaxPooling2D)  (None, 8, 8, 64)                    0

 dropout_17 (Dropout)             (None, 8, 8, 64)                    0

 conv2d_27 (Conv2D)               (None, 8, 8, 128)              73,856
 batch_normalization_14           (None, 8, 8, 128)                512
 (BatchNormalization)

 max_pooling2d_14 (MaxPooling2D)  (None, 4, 4, 128)                  0

 dropout_18 (Dropout)             (None, 4, 4, 128)                  0

 flatten_4 (Flatten)              (None, 2048)                       0

 dense_8 (Dense)                  (None, 256)                  524,544

 dropout_19 (Dropout)             (None, 256)                        0

 dense_9 (Dense)                  (None, 10)                     2,570


Total params: 667,434 (2.55 MB)


Trainable params: 666,986 (2.54 MB)


Non-trainable params: 448 (1.75 KB)
```

[49]:

```python
datagen = ImageDataGenerator(
    rotation_range=25,
    width_shift_range=0.2,
    height_shift_range=0.2,
    brightness_range=[0.8, 1.2],
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

datagen.fit(x_train)
```

[50]:

```python
batch_size = 64
epochs = 25

# Custom learning rate scheduler
def scheduler(epoch, lr):
    if epoch > 15:
        return lr * 0.5
    else:
        return lr
```

```
lr_callback = tf.keras.callbacks.LearningRateScheduler(scheduler)

# Training the model
history = model.fit(datagen.flow(x_train, y_train, batch_size=batch_size),
                    epochs=epochs,
                    validation_data=(x_test, y_test),
                    callbacks=[lr_callback])
```

```
Epoch 1/25
782/782            42s 52ms/step -
accuracy: 0.0995 - loss: 2.6447 - val_accuracy: 0.1710 - val_loss: 2.3436 -
learning_rate: 0.0010
Epoch 2/25
782/782            46s 59ms/step -
accuracy: 0.1118 - loss: 2.2922 - val_accuracy: 0.1736 - val_loss: 2.5820 -
learning_rate: 0.0010
Epoch 3/25
782/782            45s 58ms/step -
accuracy: 0.1148 - loss: 2.2811 - val_accuracy: 0.1756 - val_loss: 2.0990 -
learning_rate: 0.0010
Epoch 4/25
782/782            43s 55ms/step -
accuracy: 0.1133 - loss: 2.2828 - val_accuracy: 0.1868 - val_loss: 2.0923 -
learning_rate: 0.0010
Epoch 5/25
782/782            42s 53ms/step -
accuracy: 0.1168 - loss: 2.2736 - val_accuracy: 0.2132 - val_loss: 2.0571 -
learning_rate: 0.0010
Epoch 6/25
782/782            42s 53ms/step -
accuracy: 0.1235 - loss: 2.2711 - val_accuracy: 0.2259 - val_loss: 2.0011 -
learning_rate: 0.0010
Epoch 7/25
782/782            42s 53ms/step -
accuracy: 0.1206 - loss: 2.2704 - val_accuracy: 0.1730 - val_loss: 2.0595 -
learning_rate: 0.0010
Epoch 8/25
782/782            43s 55ms/step -
accuracy: 0.1179 - loss: 2.2699 - val_accuracy: 0.2505 - val_loss: 2.0373 -
learning_rate: 0.0010
Epoch 9/25
782/782            42s 54ms/step -
accuracy: 0.1206 - loss: 2.2673 - val_accuracy: 0.1742 - val_loss: 2.0919 -
learning_rate: 0.0010
Epoch 10/25
782/782            42s 53ms/step -
accuracy: 0.1257 - loss: 2.2665 - val_accuracy: 0.1928 - val_loss: 2.0056 -
learning_rate: 0.0010
```

```
Epoch 11/25
782/782             41s 53ms/step -
accuracy: 0.1234 - loss: 2.2643 - val_accuracy: 0.2441 - val_loss: 1.8949 -
learning_rate: 0.0010
Epoch 12/25
782/782             41s 53ms/step -
accuracy: 0.1294 - loss: 2.2523 - val_accuracy: 0.1846 - val_loss: 2.1228 -
learning_rate: 0.0010
Epoch 13/25
782/782             41s 53ms/step -
accuracy: 0.1286 - loss: 2.2560 - val_accuracy: 0.2376 - val_loss: 1.9743 -
learning_rate: 0.0010
Epoch 14/25
782/782             43s 55ms/step -
accuracy: 0.1341 - loss: 2.2506 - val_accuracy: 0.2486 - val_loss: 1.8841 -
learning_rate: 0.0010
Epoch 15/25
782/782             43s 55ms/step -
accuracy: 0.1326 - loss: 2.2460 - val_accuracy: 0.2623 - val_loss: 1.8694 -
learning_rate: 0.0010
Epoch 16/25
782/782             42s 53ms/step -
accuracy: 0.1356 - loss: 2.2428 - val_accuracy: 0.2647 - val_loss: 1.8684 -
learning_rate: 0.0010
Epoch 17/25
782/782             44s 56ms/step -
accuracy: 0.1359 - loss: 2.2327 - val_accuracy: 0.2666 - val_loss: 1.7981 -
learning_rate: 5.0000e-04
Epoch 18/25
782/782             44s 56ms/step -
accuracy: 0.1418 - loss: 2.2230 - val_accuracy: 0.2676 - val_loss: 1.8265 -
learning_rate: 2.5000e-04
Epoch 19/25
782/782             44s 56ms/step -
accuracy: 0.1451 - loss: 2.2173 - val_accuracy: 0.2812 - val_loss: 1.8177 -
learning_rate: 1.2500e-04
Epoch 20/25
782/782             44s 56ms/step -
accuracy: 0.1466 - loss: 2.2149 - val_accuracy: 0.3064 - val_loss: 1.7588 -
learning_rate: 6.2500e-05
Epoch 21/25
782/782             43s 55ms/step -
accuracy: 0.1501 - loss: 2.2066 - val_accuracy: 0.3278 - val_loss: 1.7301 -
learning_rate: 3.1250e-05
Epoch 22/25
782/782             44s 57ms/step -
accuracy: 0.1507 - loss: 2.2123 - val_accuracy: 0.3236 - val_loss: 1.7406 -
learning_rate: 1.5625e-05
```

```
Epoch 23/25
782/782              44s 56ms/step -
accuracy: 0.1479 - loss: 2.2116 - val_accuracy: 0.3258 - val_loss: 1.7405 -
learning_rate: 7.8125e-06
Epoch 24/25
782/782              44s 56ms/step -
accuracy: 0.1474 - loss: 2.2111 - val_accuracy: 0.3276 - val_loss: 1.7387 -
learning_rate: 3.9063e-06
Epoch 25/25
782/782              45s 57ms/step -
accuracy: 0.1475 - loss: 2.2097 - val_accuracy: 0.3268 - val_loss: 1.7359 -
learning_rate: 1.9531e-06
```

[51]:

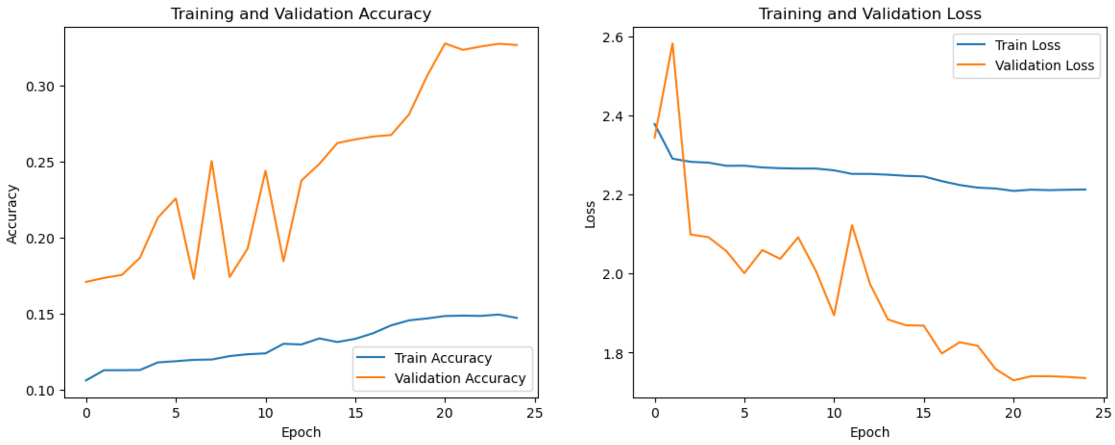
```python
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_accuracy:.4f}")
```

```
313/313 - 3s - 10ms/step - accuracy: 0.3268 - loss: 1.7359
Test accuracy: 0.3268
```

[52]:

```python
# Plot accuracy
plt.figure(figsize=(14, 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('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



```
[63]: model.save('unique_cnn_image_classification_model.keras')
```

```
[ ]:
```