3. CNN model

February 10, 2026

Author : Vaibhav Thakur

[3]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import warnings

# Suppress all warnings
warnings.filterwarnings("ignore")

# Define the paths
train_dir = '/Users/sundaramvaibhav/Documents/MRI_Dataset/train'
test_dir = '/Users/sundaramvaibhav/Documents/MRI_Dataset/test'

# Create the ImageDataGenerator object for data augmentation and preprocessing
train_datagen = ImageDataGenerator(
    rescale = 1./255,
    shear_range = 0.4,
    zoom_range = 0.2,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    rotation_range = 90,
    horizontal_flip = True,
    fill_mode = 'nearest'
)
test_datagen = ImageDataGenerator(rescale = 1./255)

# Load the images and labels, ensuring grayscale (1 color channel)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size = (128, 128),      # Resize the images to 128 x 128
    color_mode = 'grayscale',      # we have not used batch_size, so keras by␣
 ↪default chooses batch_size = 32
    class_mode = 'categorical'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size = (128,128),
```

```
    color_mode = 'grayscale',      # Load as grayscale
    class_mode = 'categorical',
)
```

```
Found 10240 images belonging to 4 classes.
Found 1279 images belonging to 4 classes.
```

CNN MODEL

[2]:
```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import Flatten
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Dropout
import numpy as np
```

[3]:
```
# Define the CNN model
CNN1 = Sequential()

# Convolutional layers
CNN1.add(Conv2D(256, (3,3), activation = 'relu', input_shape = (128, 128, 1)))
CNN1.add(MaxPooling2D(pool_size = (2,2)))

CNN1.add(Conv2D(64, (3,3), activation = 'relu'))
CNN1.add(MaxPooling2D(pool_size = (2,2)))

CNN1.add(Conv2D(256, (3,3), activation = 'relu'))
CNN1.add(MaxPooling2D(pool_size = (2,2)))

CNN1.add(Conv2D(128, (3,3), activation = 'relu'))
CNN1.add(MaxPooling2D(pool_size = (2,2)))

CNN1.add(Conv2D(256, (3,3), activation = 'relu'))
CNN1.add(MaxPooling2D(pool_size = (2,2)))

# Flattening the layers before feeding into dense layers
CNN1.add(Flatten())

# Fully connected layers
CNN1.add(Dense(128, activation = 'relu'))
CNN1.add(Dropout(0.2)) # Dropout to prevent overfitting
CNN1.add(Dense(64, activation = 'relu'))

# Output layer (adjust number of units to match the number of classes)
```

```
CNN1.add(Dense(4, activation = 'softmax')) # Assuming 4 classes based on the␣
 ↪dataset

# Compile the model
CNN1.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =␣
 ↪['accuracy'])

# Model summary
CNN1.summary()
```

```
2026-02-10 20:04:47.294911: I metal_plugin/src/device/metal_device.cc:1154]
Metal device set to: Apple M1
2026-02-10 20:04:47.294962: I metal_plugin/src/device/metal_device.cc:296]
systemMemory: 16.00 GB
2026-02-10 20:04:47.294977: I metal_plugin/src/device/metal_device.cc:313]
maxCacheSize: 5.33 GB
2026-02-10 20:04:47.295005: 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.
2026-02-10 20:04:47.295025: 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>)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 256)	2,560
max_pooling2d (MaxPooling2D)	(None, 63, 63, 256)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	147,520
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 256)	147,712
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 256)	0
conv2d_3 (Conv2D)	(None, 12, 12, 128)	295,040
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 128)	0

conv2d_4 (Conv2D)	(None, 4, 4, 256)	295,168
max_pooling2d_4 (MaxPooling2D)	(None, 2, 2, 256)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 128)	131,200
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 4)	260

Total params: 1,027,716 (3.92 MB)

Trainable params: 1,027,716 (3.92 MB)

Non-trainable params: 0 (0.00 B)

[4]:
```
CNN1.fit(train_generator, epochs = 50) # we can also defiine batch_size here
```

```
Epoch 1/50
2026-02-10 20:04:59.283021: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:117]
Plugin optimizer for device_type GPU is enabled.
320/320             81s 248ms/step -
accuracy: 0.2542 - loss: 1.3877
Epoch 2/50
320/320             78s 245ms/step -
accuracy: 0.2476 - loss: 1.3870
Epoch 3/50
320/320             78s 245ms/step -
accuracy: 0.2460 - loss: 1.3867
Epoch 4/50
320/320             80s 249ms/step -
accuracy: 0.2528 - loss: 1.3867
Epoch 5/50
320/320             81s 252ms/step -
accuracy: 0.2514 - loss: 1.3867
Epoch 6/50
320/320             81s 252ms/step -
accuracy: 0.2507 - loss: 1.3866
Epoch 7/50
```

```
320/320             79s 248ms/step -
accuracy: 0.2412 - loss: 1.3866
Epoch 8/50
320/320             80s 250ms/step -
accuracy: 0.2527 - loss: 1.3864
Epoch 9/50
320/320             83s 259ms/step -
accuracy: 0.2452 - loss: 1.3867
Epoch 10/50
320/320             89s 277ms/step -
accuracy: 0.2469 - loss: 1.3866
Epoch 11/50
320/320             84s 264ms/step -
accuracy: 0.2425 - loss: 1.3867
Epoch 12/50
320/320             84s 261ms/step -
accuracy: 0.2479 - loss: 1.3866
Epoch 13/50
320/320             84s 262ms/step -
accuracy: 0.2446 - loss: 1.3867
Epoch 14/50
320/320             84s 263ms/step -
accuracy: 0.2480 - loss: 1.3866
Epoch 15/50
320/320             84s 264ms/step -
accuracy: 0.2482 - loss: 1.3866
Epoch 16/50
320/320             85s 265ms/step -
accuracy: 0.2426 - loss: 1.3867
Epoch 17/50
320/320             85s 265ms/step -
accuracy: 0.2459 - loss: 1.3867
Epoch 18/50
320/320             86s 267ms/step -
accuracy: 0.2512 - loss: 1.3866
Epoch 19/50
320/320             86s 269ms/step -
accuracy: 0.2457 - loss: 1.3867
Epoch 20/50
320/320             85s 264ms/step -
accuracy: 0.2438 - loss: 1.3867
Epoch 21/50
320/320             83s 260ms/step -
accuracy: 0.2470 - loss: 1.3867
Epoch 22/50
320/320             83s 260ms/step -
accuracy: 0.2507 - loss: 1.3866
Epoch 23/50
```

```
320/320             83s 260ms/step -
accuracy: 0.2495 - loss: 1.3865
Epoch 24/50
320/320             84s 261ms/step -
accuracy: 0.2489 - loss: 1.3866
Epoch 25/50
320/320             86s 270ms/step -
accuracy: 0.2458 - loss: 1.3867
Epoch 26/50
320/320             86s 268ms/step -
accuracy: 0.2444 - loss: 1.3867
Epoch 27/50
320/320             86s 268ms/step -
accuracy: 0.2450 - loss: 1.3867
Epoch 28/50
320/320             86s 269ms/step -
accuracy: 0.2427 - loss: 1.3867
Epoch 29/50
320/320             92s 287ms/step -
accuracy: 0.2390 - loss: 1.3867
Epoch 30/50
320/320             83s 260ms/step -
accuracy: 0.2399 - loss: 1.3867
Epoch 31/50
320/320             83s 259ms/step -
accuracy: 0.2511 - loss: 1.3865
Epoch 32/50
320/320             83s 260ms/step -
accuracy: 0.2465 - loss: 1.3867
Epoch 33/50
320/320             83s 260ms/step -
accuracy: 0.2488 - loss: 1.3865
Epoch 34/50
320/320             83s 261ms/step -
accuracy: 0.2440 - loss: 1.3867
Epoch 35/50
320/320             84s 261ms/step -
accuracy: 0.2454 - loss: 1.3866
Epoch 36/50
320/320             84s 262ms/step -
accuracy: 0.2442 - loss: 1.3868
Epoch 37/50
320/320             84s 262ms/step -
accuracy: 0.2463 - loss: 1.3868
Epoch 38/50
320/320             84s 263ms/step -
accuracy: 0.2413 - loss: 1.3867
Epoch 39/50
```

```
320/320             84s 263ms/step -
accuracy: 0.2427 - loss: 1.3867
Epoch 40/50
320/320             85s 264ms/step -
accuracy: 0.2498 - loss: 1.3866
Epoch 41/50
320/320             84s 263ms/step -
accuracy: 0.2388 - loss: 1.3868
Epoch 42/50
320/320             85s 265ms/step -
accuracy: 0.2396 - loss: 1.3867
Epoch 43/50
320/320             85s 265ms/step -
accuracy: 0.2437 - loss: 1.3866
Epoch 44/50
320/320             85s 266ms/step -
accuracy: 0.2470 - loss: 1.3867
Epoch 45/50
320/320             85s 266ms/step -
accuracy: 0.2434 - loss: 1.3866
Epoch 46/50
320/320             86s 268ms/step -
accuracy: 0.2492 - loss: 1.3866
Epoch 47/50
320/320             87s 272ms/step -
accuracy: 0.2395 - loss: 1.3867
Epoch 48/50
320/320             86s 270ms/step -
accuracy: 0.2513 - loss: 1.3866
Epoch 49/50
320/320             86s 269ms/step -
accuracy: 0.2495 - loss: 1.3866
Epoch 50/50
320/320             87s 271ms/step -
accuracy: 0.2456 - loss: 1.3866
```

[4]:
```
<keras.src.callbacks.history.History at 0x319729f30>
```

[6]:
```
# Evaluate the model on the test data
loss, accuracy1 = CNN1.evaluate(test_generator)

# Print the accuracy
print(f"Test accuracy of CNN1: {accuracy1}")
```

```
40/40             2s 53ms/step -
accuracy: 0.5004 - loss: 1.3803
Test accuracy of CNN1: 0.5003909468650818
```

With Batch Normalization

[8]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout, BatchNormalization

# Define the CNN model
CNN1 = Sequential()

# Convolutional Block 1
CNN1.add(Conv2D(256, (3,3), activation = 'relu', input_shape = (128, 128, 1)))
CNN1.add(BatchNormalization()) # Added Batch Normalization
CNN1.add(MaxPooling2D(pool_size = (2,2)))

# Convolutional Block 2
CNN1.add(Conv2D(64, (3,3), activation = 'relu', input_shape = (128, 128, 1)))
CNN1.add(BatchNormalization()) # Added Batch Normalization
CNN1.add(MaxPooling2D(pool_size = (2,2)))

# Convolutional Block 3
CNN1.add(Conv2D(256, (3,3), activation = 'relu', input_shape = (128, 128, 1)))
CNN1.add(BatchNormalization()) # Added Batch Normalization
CNN1.add(MaxPooling2D(pool_size = (2,2)))

# Convolutional Block 4
CNN1.add(Conv2D(128, (3,3), activation = 'relu', input_shape = (128, 128, 1)))
CNN1.add(BatchNormalization()) # Added Batch Normalization
CNN1.add(MaxPooling2D(pool_size = (2,2)))

# Convolutional Block 5
CNN1.add(Conv2D(256, (3,3), activation = 'relu', input_shape = (128, 128, 1)))
CNN1.add(BatchNormalization()) # Added Batch Normalization
CNN1.add(MaxPooling2D(pool_size = (2,2)))

# Flatten before Dense layers
CNN1.add(Flatten())

# Fully Connected Layers
CNN1.add(Dense(128, activation = 'relu'))
CNN1.add(BatchNormalization())  # Optional BN before dropout
CNN1.add(Dropout(0.2))

CNN1.add(Dense(64, activation = 'relu'))
CNN1.add(BatchNormalization())  # Optional BN before dropout
CNN1.add(Dropout(0.2))

# Output Layer
```

```
CNN1.add(Dense(4, activation = 'softmax')) # Assuming 4 classes

# Compile the model
CNN1.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =␣
 ↪['accuracy'])

# Model summary
CNN1.summary()
```

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 126, 126, 256)	2,560
batch_normalization (BatchNormalization)	(None, 126, 126, 256)	1,024
max_pooling2d_5 (MaxPooling2D)	(None, 63, 63, 256)	0
conv2d_7 (Conv2D)	(None, 61, 61, 64)	147,520
batch_normalization_1 (BatchNormalization)	(None, 61, 61, 64)	256
max_pooling2d_6 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_8 (Conv2D)	(None, 28, 28, 256)	147,712
batch_normalization_2 (BatchNormalization)	(None, 28, 28, 256)	1,024
max_pooling2d_7 (MaxPooling2D)	(None, 14, 14, 256)	0
conv2d_9 (Conv2D)	(None, 12, 12, 128)	295,040
batch_normalization_3 (BatchNormalization)	(None, 12, 12, 128)	512
max_pooling2d_8 (MaxPooling2D)	(None, 6, 6, 128)	0
conv2d_10 (Conv2D)	(None, 4, 4, 256)	295,168
batch_normalization_4 (BatchNormalization)	(None, 4, 4, 256)	1,024

max_pooling2d_9 (MaxPooling2D)	(None, 2, 2, 256)	0
flatten_1 (Flatten)	(None, 1024)	0
dense_3 (Dense)	(None, 128)	131,200
batch_normalization_5 (BatchNormalization)	(None, 128)	512
dropout_1 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 64)	8,256
batch_normalization_6 (BatchNormalization)	(None, 64)	256
dropout_2 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 4)	260

Total params: 1,032,324 (3.94 MB)

Trainable params: 1,030,020 (3.93 MB)

Non-trainable params: 2,304 (9.00 KB)

[9]:
```
CNN1.fit(train_generator, epochs = 2)
# Only 2 epochs to save time and compute
```

```
Epoch 1/2
320/320              188s 575ms/step -
accuracy: 0.4096 - loss: 1.3547
Epoch 2/2
320/320              190s 595ms/step -
accuracy: 0.5792 - loss: 0.9806
```

[9]: <keras.src.callbacks.history.History at 0x371f69090>

[10]:
```
# Evaluate the model on the test data
loss, accuracy1 = CNN1.evaluate(test_generator)

# Print the accuracy
print(f"Test accuracy of CNN1 with batch normalization: {accuracy1}")
```

```
40/40              4s 92ms/step -
accuracy: 0.3503 - loss: 1.6848
Test accuracy of CNN1 with batch normalization: 0.35027363896369934
```

With L1 regularization

[1]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, BatchNormalization
from tensorflow.keras import regularizers

# Define the CNN model
CNN1 = Sequential()

# Convolutional layers with L1 regularization
CNN1.add(Conv2D(256, (3, 3), activation = 'relu', input_shape = (128, 128, 1),
               kernel_regularizer = regularizers.l1(0.001)))
CNN1.add(MaxPooling2D(pool_size = (2,2)))

CNN1.add(Conv2D(64, (3, 3), activation = 'relu', input_shape = (128, 128, 1),
               kernel_regularizer = regularizers.l1(0.001)))
CNN1.add(MaxPooling2D(pool_size = (2,2)))

CNN1.add(Conv2D(256, (3, 3), activation = 'relu', input_shape = (128, 128, 1),
               kernel_regularizer = regularizers.l1(0.001)))
CNN1.add(MaxPooling2D(pool_size = (2,2)))

CNN1.add(Conv2D(128, (3, 3), activation = 'relu', input_shape = (128, 128, 1),
               kernel_regularizer = regularizers.l1(0.001)))
CNN1.add(MaxPooling2D(pool_size = (2,2)))

CNN1.add(Conv2D(256, (3, 3), activation = 'relu', input_shape = (128, 128, 1),
               kernel_regularizer = regularizers.l1(0.001)))
CNN1.add(MaxPooling2D(pool_size = (2,2)))

# Flatten before Dense layers
CNN1.add(Flatten())

# Fully Connected Layers with L1 regularization
CNN1.add(Dense(128, activation = 'relu',
              kernel_regularizer = regularizers.l1(0.001)))
CNN1.add(Dropout(0.2))

CNN1.add(Dense(64, activation = 'relu',
              kernel_regularizer = regularizers.l1(0.001)))
CNN1.add(Dropout(0.2))

# Output Layer
```

```
CNN1.add(Dense(4, activation = 'softmax')) # Assuming 4 classes

# Compile the model
CNN1.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =␣
 ↪['accuracy'])

# Model summary
CNN1.summary()
```

```
2026-02-10 22:41:24.149721: I metal_plugin/src/device/metal_device.cc:1154]
Metal device set to: Apple M1
2026-02-10 22:41:24.149743: I metal_plugin/src/device/metal_device.cc:296]
systemMemory: 16.00 GB
2026-02-10 22:41:24.149748: I metal_plugin/src/device/metal_device.cc:313]
maxCacheSize: 5.33 GB
2026-02-10 22:41:24.149767: 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.
2026-02-10 22:41:24.149776: 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>)
/opt/anaconda3/envs/py310/lib/python3.10/site-
packages/keras/src/layers/convolutional/base_conv.py:113: 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)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 256)	2,560
max_pooling2d (MaxPooling2D)	(None, 63, 63, 256)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	147,520
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 256)	147,712
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 256)	0

```
 conv2d_3 (Conv2D)               (None, 12, 12, 128)          295,040

 max_pooling2d_3 (MaxPooling2D)  (None, 6, 6, 128)                 0

 conv2d_4 (Conv2D)               (None, 4, 4, 256)            295,168

 max_pooling2d_4 (MaxPooling2D)  (None, 2, 2, 256)                 0

 flatten (Flatten)               (None, 1024)                      0

 dense (Dense)                   (None, 128)                  131,200

 dropout (Dropout)               (None, 128)                       0

 dense_1 (Dense)                 (None, 64)                     8,256

 dropout_1 (Dropout)             (None, 64)                        0

 dense_2 (Dense)                 (None, 4)                       260


Total params: 1,027,716 (3.92 MB)

Trainable params: 1,027,716 (3.92 MB)

Non-trainable params: 0 (0.00 B)
```

[4]:
```
CNN1.fit(train_generator, epochs = 2)
```

```
Epoch 1/2
2026-02-10 22:42:11.506549: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:117]
Plugin optimizer for device_type GPU is enabled.
320/320            79s 244ms/step -
accuracy: 0.2485 - loss: 3.0018
Epoch 2/2
320/320            81s 253ms/step -
accuracy: 0.2433 - loss: 1.5129
```

[4]:
```
<keras.src.callbacks.history.History at 0x30d16c700>
```

[5]:
```
# Evaluate the model on the test data
loss, accuracy1 = CNN1.evaluate(test_generator)

# Print the accuracy
```

```
print(f"Test accuracy of CNN1 with L1 regularization: {accuracy1}")
```

```
40/40             2s 49ms/step -
accuracy: 0.0094 - loss: 1.5146
Test accuracy of CNN1 with L1 regularization: 0.009382329881191254
```

For L2, replace l1 with l2 in the above code

With Batch regularization, L2 regualrization, Dropout

[9]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, BatchNormalization, Activation
from tensorflow.keras import regularizers
from tensorflow.keras.optimizers import Adam

CNN1 = Sequential()


# - - - - Block 1 - - - -
CNN1.add(Conv2D(256, (3, 3), input_shape = (128, 128, 1),
                kernel_regularizer = regularizers.l2(0.001)))
CNN1.add(BatchNormalization())
CNN1.add(Activation('relu'))
CNN1.add(MaxPooling2D(pool_size = (2,2)))
CNN1.add(Dropout(0.25)) # small dropout after conv/pool (optional)


# - - - - Block 2 - - - -
CNN1.add(Conv2D(64, (3, 3), input_shape = (128, 128, 1),
                kernel_regularizer = regularizers.l2(0.001)))
CNN1.add(BatchNormalization())
CNN1.add(Activation('relu'))
CNN1.add(MaxPooling2D(pool_size = (2,2)))
CNN1.add(Dropout(0.25))


# - - - - Block 3 - - - -
CNN1.add(Conv2D(256, (3, 3), input_shape = (128, 128, 1),
                kernel_regularizer = regularizers.l2(0.001)))
CNN1.add(BatchNormalization())
CNN1.add(Activation('relu'))
CNN1.add(MaxPooling2D(pool_size = (2,2)))
CNN1.add(Dropout(0.25))

# - - - - Block 4 - - - -
CNN1.add(Conv2D(128, (3, 3), input_shape = (128, 128, 1),
```

```
                kernel_regularizer = regularizers.l2(0.001)))
CNN1.add(BatchNormalization())
CNN1.add(Activation('relu'))
CNN1.add(MaxPooling2D(pool_size = (2,2)))
CNN1.add(Dropout(0.25))

# - - - - Block 5 - - - -
CNN1.add(Conv2D(256, (3, 3), input_shape = (128, 128, 1),
                kernel_regularizer = regularizers.l2(0.001)))
CNN1.add(BatchNormalization())
CNN1.add(Activation('relu'))
CNN1.add(MaxPooling2D(pool_size = (2,2)))
CNN1.add(Dropout(0.25))

# - - - - Classifier - - - -
CNN1.add(Flatten())

CNN1.add(Dense(128, kernel_regularizer = regularizers.l2(0.001)))
CNN1.add(BatchNormalization())
CNN1.add(Activation('relu'))
CNN1.add(Dropout(0.5))  # Higher dropout on dense layers

CNN1.add(Dense(64, kernel_regularizer = regularizers.l2(0.001)))
CNN1.add(BatchNormalization())
CNN1.add(Activation('relu'))
CNN1.add(Dropout(0.5))

CNN1.add(Dense(4, activation = 'softmax'))

# Compile with slightly lower LR (pairs well with droout/BN)
CNN1.compile(optimizer = Adam(learning_rate = 0.0004),
             loss = 'categorical_crossentropy',
             metrics = ['accuracy'])

CNN1.summary()
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
conv2d_15 (Conv2D)	(None, 126, 126, 256)	2,560
batch_normalization_12 (BatchNormalization)	(None, 126, 126, 256)	1,024
activation_10 (Activation)	(None, 126, 126, 256)	0

```
max_pooling2d_15 (MaxPooling2D)   (None, 63, 63, 256)              0

dropout_12 (Dropout)              (None, 63, 63, 256)              0

conv2d_16 (Conv2D)                (None, 61, 61, 64)           147,520

batch_normalization_13            (None, 61, 61, 64)               256
(BatchNormalization)

activation_11 (Activation)        (None, 61, 61, 64)               0

max_pooling2d_16 (MaxPooling2D)   (None, 30, 30, 64)               0

dropout_13 (Dropout)              (None, 30, 30, 64)               0

conv2d_17 (Conv2D)                (None, 28, 28, 256)          147,712

batch_normalization_14            (None, 28, 28, 256)            1,024
(BatchNormalization)

activation_12 (Activation)        (None, 28, 28, 256)              0

max_pooling2d_17 (MaxPooling2D)   (None, 14, 14, 256)              0

dropout_14 (Dropout)              (None, 14, 14, 256)              0

conv2d_18 (Conv2D)                (None, 12, 12, 128)          295,040

batch_normalization_15            (None, 12, 12, 128)              512
(BatchNormalization)

activation_13 (Activation)        (None, 12, 12, 128)              0

max_pooling2d_18 (MaxPooling2D)   (None, 6, 6, 128)                0

dropout_15 (Dropout)              (None, 6, 6, 128)                0

conv2d_19 (Conv2D)                (None, 4, 4, 256)            295,168

batch_normalization_16            (None, 4, 4, 256)              1,024
(BatchNormalization)

activation_14 (Activation)        (None, 4, 4, 256)                0

max_pooling2d_19 (MaxPooling2D)   (None, 2, 2, 256)                0

dropout_16 (Dropout)              (None, 2, 2, 256)                0
```

flatten_3 (Flatten)	(None, 1024)	0
dense_5 (Dense)	(None, 128)	131,200
batch_normalization_17 (BatchNormalization)	(None, 128)	512
activation_15 (Activation)	(None, 128)	0
dropout_17 (Dropout)	(None, 128)	0
dense_6 (Dense)	(None, 64)	8,256
batch_normalization_18 (BatchNormalization)	(None, 64)	256
activation_16 (Activation)	(None, 64)	0
dropout_18 (Dropout)	(None, 64)	0
dense_7 (Dense)	(None, 4)	260

Total params: 1,032,324 (3.94 MB)

Trainable params: 1,030,020 (3.93 MB)

Non-trainable params: 2,304 (9.00 KB)

[10]:
```
CNN1.fit(train_generator, epochs = 2)
```

```
Epoch 1/2
320/320             201s 618ms/step -
accuracy: 0.2828 - loss: 2.5458
Epoch 2/2
320/320             202s 632ms/step -
accuracy: 0.3534 - loss: 2.2521
```

[10]: <keras.src.callbacks.history.History at 0x36cf2bc70>

[11]:
```
# Evaluate the model on the test data
loss, accuracy1 = CNN1.evaluate(test_generator)

# Print the accuracy
print(f"Test accuracy of CNN1 model: {accuracy1}")
```

```
40/40              4s 95ms/step -
accuracy: 0.5059 - loss: 1.8347
Test accuracy of CNN1 model: 0.5058639645576477
```

[ ]: