CNNモデル構築コードの行ごとの解説

📌 モデルの構築

```
# Build a simple CNN model
```

- 単純なCNNモデルの構築を開始。

```
model = models.Sequential([
```

- モデルを`Sequential`で定義。レイヤーを順番に積み重ねる構造。

```
    tf.keras.Input(shape=(224, 224, 3)),
```

- 入力画像の形を指定（224×224ピクセル、3チャンネルRGB）。

```
    layers.Rescaling(1./255),
```

- ピクセル値を0〜1に正規化。

◇ Block 1（32チャンネル）

```
    layers.Conv2D(32, (3, 3), padding='same', activation='relu',
                  kernel_regularizer=regularizers.l2(1e-4)),
```

- 3×3の畳み込み層（32フィルタ）、ReLU活性化関数、L2正則化。

```
    layers.BatchNormalization(),
```

- バッチ正規化で学習を安定化。

```
    layers.Conv2D(32, (3, 3), padding='same', activation='relu',
                  kernel_regularizer=regularizers.l2(1e-4)),
```

- 同じ設定でもう1層のConv。

```
    layers.MaxPooling2D(),
```

- 空間サイズを半分に縮小。

◇ Block 2（64チャンネル）

```
    layers.Conv2D(64, (3, 3), padding='same', activation='relu',
                  kernel_regularizer=regularizers.l2(1e-4)),
    layers.BatchNormalization(),
    layers.Conv2D(64, (3, 3), padding='same', activation='relu',
                  kernel_regularizer=regularizers.l2(1e-4)),
    layers.MaxPooling2D(),
```

- 同様の構造で、フィルター数を64に増加。

◇ Block 3（128チャンネル）

```
    layers.Conv2D(128, (3, 3), padding='same', activation='relu',
                  kernel_regularizer=regularizers.l2(1e-4)),
    layers.BatchNormalization(),
    layers.Conv2D(128, (3, 3), padding='same', activation='relu',
                  kernel_regularizer=regularizers.l2(1e-4)),
    layers.MaxPooling2D(),
```

- フィルター数をさらに増やして特徴量の抽出能力を高める。

◇ Block 4（256チャンネル）

```
    layers.Conv2D(256, (3, 3), padding='same', activation='relu',
                  kernel_regularizer=regularizers.l2(1e-4)),
    layers.BatchNormalization(),
    layers.MaxPooling2D(),
```

- 畳み込み1層で終了、より深い特徴抽出。

◇ 全結合層への接続とDropout

```
    layers.Flatten(),
```

- 多次元の特徴マップを1次元に変換。

```
    layers.Dropout(0.3),
```

- 過学習防止のためのDropout（30%無効化）。

◇ 全結合層（Dense）

```
    layers.Dense(512, activation='relu', kernel_regularizer=regularizers.l2(1e-
4)),
    layers.Dropout(0.25),
    layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l2(1e-
4)),
    layers.Dropout(0.25),
```

- 中間表現を学習するDense層とDropoutで正則化。

```
    layers.Dense(num_classes, activation='softmax')
```

- 最終層：クラス数分のノードを持つソフトマックス分類層。

🔧 モデルのコンパイル

```
model.compile(
    optimizer=Adam(learning_rate=1e-4),
```

- 学習率`1e-4`のAdam最適化器で最適化。

```
    loss='categorical_crossentropy',
```

- 複数クラス分類のための損失関数。

```
    metrics=['accuracy']
)
```

- 精度を評価指標に設定。