In [1]:
```
#Problem 1 a)i)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D

# Create the model
model = Sequential()

# Add the first 2D convolutional layer:
# - 64 filters
# - Kernel size of (5, 5)
# - ReLU activation function
# - Input shape matching CIFAR-10 images (32x32 pixels with 3 channels)
model.add(Conv2D(64, (5, 5), activation='relu', input_shape=(32, 32, 3)))

# Print model summary to verify the layer configuration
model.summary()
```

```
C:\Users\Tom\anaconda3\Lib\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)
```

Model: "sequential"

Layer (type)	Output Shape
conv2d (Conv2D)	(None, 28, 28, 64)

Total params: 4,864 (19.00 KB)

Trainable params: 4,864 (19.00 KB)

Non-trainable params: 0 (0.00 B)

In [2]:
```
#Problem 1 a)ii)
from tensorflow.keras.layers import MaxPooling2D

# Add the second layer: a max pooling layer with pool size of (2, 2)
model.add(MaxPooling2D(pool_size=(2, 2)))

# Optionally, you can print the model summary to verify the addition
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape
conv2d (Conv2D)	(None, 28, 28, 64)
max_pooling2d (MaxPooling2D)	(None, 14, 14, 64)

Total params: 4,864 (19.00 KB)

Trainable params: 4,864 (19.00 KB)

Non-trainable params: 0 (0.00 B)

In [3]:
```python
#Problem 1 a the rest
from tensorflow.keras.layers import Flatten, Dense

# Create the model
model = Sequential()

# i. First layer: 2D convolution with 64 filters, (5,5) kernel, ReLU activat
#    Input shape is set to (32, 32, 3) as CIFAR-10 images are 32x32 RGB imag
model.add(Conv2D(64, (5, 5), activation='relu', input_shape=(32, 32, 3)))

# ii. Second layer: Max pooling with pool size (2,2)
model.add(MaxPooling2D(pool_size=(2, 2)))

# iii. Third layer: 2D convolution with 32 filters, (3,3) kernel, ReLU activ
model.add(Conv2D(32, (3, 3), activation='relu'))

# iv. Fourth layer: Another max pooling layer with pool size (2,2)
model.add(MaxPooling2D(pool_size=(2, 2)))

# v. Fifth layer: Another 2D convolution with 32 filters, (3,3) kernel, ReLU
model.add(Conv2D(32, (3, 3), activation='relu'))

# vi. Sixth layer: Flatten layer to convert 2D output into a 1D vector.
model.add(Flatten())

# vii. Seventh layer: Dense layer with 64 neurons and ReLU activation.
model.add(Dense(64, activation='relu'))

# viii. Eighth (final) layer: Dense layer to produce probabilities for CIFAR
#      We use softmax activation so that the outputs sum to 1.
model.add(Dense(10, activation='softmax'))

# Print the model summary to verify the architecture
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape
conv2d_1 (Conv2D)	(None, 28, 28, 64)
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 64)
conv2d_2 (Conv2D)	(None, 12, 12, 32)
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 32)
conv2d_3 (Conv2D)	(None, 4, 4, 32)
flatten (Flatten)	(None, 512)
dense (Dense)	(None, 64)
dense_1 (Dense)	(None, 10)

Total params: 66,058 (258.04 KB)

Trainable params: 66,058 (258.04 KB)

Non-trainable params: 0 (0.00 B)

In [4]:
```
#Problem 2
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the dataset
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)

# Select only the three required features:
# "worst area", "worst compactness", "worst concavity"
X = df[['worst area', 'worst compactness', 'worst concavity']].values
y = data.target

# Split the data (e.g., 80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# Scale features for SVM
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# (a) SVM with a linear kernel
svm_linear = SVC(kernel='linear', random_state=42)
svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)
accuracy_linear = accuracy_score(y_test, y_pred_linear)
print("Accuracy with linear kernel: {:.4f}".format(accuracy_linear))

# (b) SVM with an RBF kernel and regularization parameter C=2
svm_rbf = SVC(kernel='rbf', C=2, random_state=42)
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)
print("Accuracy with RBF kernel (C=2): {:.4f}".format(accuracy_rbf))
```

```
Accuracy with linear kernel: 0.9912
Accuracy with RBF kernel (C=2): 0.9649
```

In [ ]:
```
```