Deep Learning Exam - Task Solutions

# Task 1: Image Classification - CNN from Scratch

In this task, we implement a Convolutional Neural Network (CNN) from scratch to classify images of vehicles into different categories.   
CNNs are widely used in image classification because they automatically learn spatial hierarchies of features through layers of convolutions, pooling, and fully connected layers.   
We use data augmentation to prevent overfitting and improve the model's ability to generalize to unseen images.

Code:

import tensorflow as tf  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout  
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
  
# CNN Model from scratch  
model = Sequential([  
 Conv2D(32, (3, 3), activation='relu', input\_shape=(150, 150, 3)),  
 MaxPooling2D(2, 2),  
 Conv2D(64, (3, 3), activation='relu'),  
 MaxPooling2D(2, 2),  
 Conv2D(128, (3, 3), activation='relu'),  
 MaxPooling2D(2, 2),  
 Flatten(),  
 Dense(512, activation='relu'),  
 Dropout(0.5),  
 Dense(3, activation='softmax') # Assuming 3 categories of vehicles  
])  
  
# Compile the model  
model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  
  
# Data Augmentation for the training dataset  
train\_datagen = ImageDataGenerator(rescale=1./255, rotation\_range=40, width\_shift\_range=0.2,  
 height\_shift\_range=0.2, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True)  
train\_generator = train\_datagen.flow\_from\_directory('data/train', target\_size=(150, 150), batch\_size=20, class\_mode='categorical')  
  
# Train the model  
history = model.fit(train\_generator, epochs=10)

# Task 2: Sentiment Analysis - RNN vs LSTM

Sentiment analysis is the process of classifying text data, such as movie reviews, into positive or negative sentiments.   
In this task, we compare the performance of a basic Recurrent Neural Network (RNN) and a Long Short-Term Memory (LSTM) network.   
LSTMs are an extension of RNNs that are better at handling long-term dependencies in text sequences.

Code:

import tensorflow as tf  
from tensorflow.keras.preprocessing.text import Tokenizer  
from tensorflow.keras.preprocessing.sequence import pad\_sequences  
from tensorflow.keras.layers import Embedding, SimpleRNN, LSTM, Dense  
from tensorflow.keras.models import Sequential  
  
# Sample movie review data (placeholder)  
sentences = ['The movie was fantastic!', 'I did not like the movie.', 'It was okay.']  
labels = [1, 0, 1] # 1 = Positive, 0 = Negative  
  
# Tokenization and padding  
tokenizer = Tokenizer(num\_words=5000)  
tokenizer.fit\_on\_texts(sentences)  
sequences = tokenizer.texts\_to\_sequences(sentences)  
padded\_sequences = pad\_sequences(sequences, maxlen=100)  
  
# Build RNN model  
rnn\_model = Sequential([  
 Embedding(5000, 128, input\_length=100),  
 SimpleRNN(128, activation='relu'),  
 Dense(1, activation='sigmoid')  
])  
  
rnn\_model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  
  
# Build LSTM model  
lstm\_model = Sequential([  
 Embedding(5000, 128, input\_length=100),  
 LSTM(128, activation='relu'),  
 Dense(1, activation='sigmoid')  
])  
  
lstm\_model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Task 3: Image Segmentation - U-Net for Medical Images

Image segmentation is the process of labeling pixels in an image, often to identify objects or boundaries within an image.   
In this task, we implement the U-Net architecture, which is widely used for medical image segmentation.   
The U-Net model features a contracting path to capture context and a symmetric expanding path to enable precise localization.

Code:

import tensorflow as tf  
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Conv2DTranspose, concatenate, Input  
from tensorflow.keras.models import Model  
  
# U-Net architecture  
def unet\_model(input\_size=(128, 128, 1)):  
 inputs = Input(input\_size)  
  
 # Contracting path  
 c1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)  
 p1 = MaxPooling2D((2, 2))(c1)  
  
 # Expanding path  
 u1 = Conv2DTranspose(64, (3, 3), strides=(2, 2), padding='same')(p1)  
 u1 = concatenate([u1, c1])  
 outputs = Conv2D(1, (1, 1), activation='sigmoid')(u1)  
  
 return Model(inputs=[inputs], outputs=[outputs])  
  
# Create the U-Net model  
model = unet\_model()  
  
# Compile the model  
model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  
  
# Load medical images for segmentation and train the model (code for loading and training goes here)