Homework 4
Deep learning

Baktash Ansari
99521082

Q1)

a)

- **Keras Tuner:**
  The Keras Tuner is a library that helps you pick the optimal set of hyperparameters for your TensorFlow program. The process of selecting the right set of hyperparameters for your machine learning (ML) application is called hyperparameter tuning or hyper tuning.

  Hyperparameters are the variables that govern the training process and the topology of an ML model. These variables remain constant over the training process and directly impact the performance of your ML program. Hyperparameters are of two types:

  Model hyperparameters that influence model selection such as the number and width of hidden layers
  Algorithm hyperparameters that influence the speed and quality of the learning algorithm such as the learning rate for Stochastic Gradient Descent (SGD) and the number of nearest neighbors for a k Nearest Neighbors (KNN) classifier.

- For using of Keras tuner first, we need to explain two things:
    1. By using Tuner we can give the range of numbers for each hyperparameter to find the best number.
       For example, for the number of units in fully connected layers, we can give a range and the tuner can find the best number of layers.
       We can do this approach by using *hp.INT.* For example, **hp.Int('units', min_value=32, max_value=512, step=32)** defines a hyperparameter named **'units'** that can take on values from 32 to 512, inclusive, with a step size of 32.

    2. We can use **Tuner**. In the context of Keras Tuner, a tuner is an object that carries out the hyperparameter tuning. The tuner object is initialized with the model-building function, the name of the objective to optimize (such as validation accuracy or loss), and the total number of trials (or different sets of hyperparameters) to test. After initialization, the search method is called on the tuner object to start the hyperparameter tuning process. The tuner explores different hyperparameters in the search space to find the combination that results in the best performance for the given objective.

- **Tuner:**
  By continuing the explanation of tuner we can use three approaches for selecting the best set of hyperparameters:
  - **Hyperband:** This tuner is based on the Hyperband algorithm, which is an efficient random search algorithm. It uses early stopping to abandon low-performing configurations, thus saving computational resources. This makes it a good choice when you have a large search space and limited resources.
  - **RandomSearch:** This tuner randomly selects combinations of hyperparameters to test. This can be a good choice when your search space is relatively small, or when you have no prior knowledge about the best hyperparameters.
  - **BayesianOptimization:** This tuner uses Bayesian inference to select hyperparameters. It's a good choice when you have some prior knowledge about the best hyperparameters, as it can use this information to make more informed decisions.

  In practice, many practitioners start with RandomSearch or Hyperband due to their efficiency and ease of use and then move to more sophisticated methods like BayesianOptimization if needed. So, for start I use RandomSearch.

b)

- **MNIST:**
  The MNIST (Modified National Institute of Standards and Technology) dataset is a large collection of handwritten digits. It is a subset of a larger dataset available at the National Institute of Standards and Technology. The MNIST database contains handwritten digits from 0 through 9 and provides a baseline for testing image processing systems. The dataset has a training set of 60,000 examples and a test set of 10,000 examples. It is a subset of a larger NIST Special Database 3 (digits written by employees of the United States Census Bureau) and Special

Database 1 (digits written by high school students) which contain monochrome images of handwritten digits. The digits have been size-normalized and centered in a fixed-size image. The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. The images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field. The MNIST dataset is one of the most researched datasets in machine learning and is often used for benchmarking machine learning algorithms. It is particularly used to classify handwritten digits.

- As I explained before we can use hp.INT for all numeric values and tuner algorithm. We can set the range for the number of layers for conv and FL layers. Also, use a numeric range for units and learning rates.

- **Pooling Layers:** Pooling layers are used in CNNs to reduce the spatial dimensions (width and height) of the input volume or feature map while preserving the depth. This is done to reduce the number of parameters and computation in the network, thereby controlling overfitting. It also makes the network invariant to small translations, rotations, and other minor changes to the input image. There are different types of pooling such as max pooling and average pooling. Max pooling selects the maximum element from the region of the feature map covered by the filter, thus preserving the most prominent features. Average pooling, on the other hand, computes the average of the elements present in the region of the feature map covered by the filter.

Dropout Layers: Dropout is a regularization technique used to prevent overfitting in neural networks. During training, dropout randomly shuts down a fraction of neurons in a layer, which helps the network to learn more robust features. This is because it forces the network to learn the features in a distributed way and not rely on any one neuron. As a result, the network becomes less sensitive to specific weights, and this leads to a more generalized model. However, dropout roughly doubles the number of iterations required to converge, but each epoch's training time is less.

Incorporating both pooling and dropout layers in your CNN can help improve the model's performance by reducing overfitting and making the model more robust to variations in the input data. However, it's important to note that the effectiveness of these techniques can vary depending on the specific task and dataset, and they should be used judiciously based on the problem at hand.

- Q1 codes are available in the notebook.

Q2) Everything is clearly visible in the notebook.

Q3)

1. One of the problems that I think we should fix in the code is that the input shape of the LSTM layer should be in this format shape: `(num_timesteps, num_features)`
   But in the code, we have: input_shape = (X_train.shape[1], 1)
   It should be (1, X_train.shape[1])
2. Another One is the input range of each value. For LSTM models, data often needs to be scaled. Indeed, we can normalize our input data.

For instance, one of our input values is the year 2023 and another one is month 1-12.

Q4)

a)

Convolutional Neural Networks:

- CNNs are a specialized type of neural network that performs convolutions over input data using learned convolutional filters to extract higher-level features. They are commonly used for image processing and computer vision tasks.
- **Use Cases:** Because they excel at finding spatial patterns, CNNs are widely used for image and video recognition, classification and segmentation. Common applications include facial recognition, medical image analysis, autonomous vehicles, and object detection/tracking.
- The convolutional and pooling layers allow CNNs to recognize spatial patterns in data while being invariant to scale, rotation, and other transformations. Their hierarchical architecture also mirrors visual perception systems in living organisms.

Recurrent Neural Networks:

- Definition: RNNs are a type of neural network well-suited for processing sequential data, with connections between nodes forming directed cycles to allow information persistence across time steps.
- **Use Cases:** RNNs are often used for natural language processing, speech recognition, and time series forecasting. The recurrent connections allow RNNs to store representations of recent input events in memory and apply this context to current data.
- The recurrent connections enable RNNs to model temporal context and dynamics, which is crucial for data like text and speech which have strong sequential correlations. Their memory also suits them for forecasting and prediction tasks.

In summary, CNNs excel at finding spatial patterns while RNNs excel at modeling temporal sequences. The architectures match the underlying structure of the data in vision (arrays) versus language/time (sequences).

b)

CNNs and RNNs differ significantly in terms of the number of parameters and ability to parallelize:

Number of Parameters:

- CNNs tend to have a very large number of parameters due to having many convolutional filters in each layer. The number of parameters grows quickly as network size and depth increase.
- RNNs generally have fewer parameters overall than similarly-sized CNNs. The recurrent connections allow RNNs to build an internal state representation over time with fewer parameters.

Parallelization:

- CNN computations intrinsically lend themselves very well to parallelization across GPUs or other hardware. The convolutions applied to each input region are independent of one another.
- RNNs are difficult to parallelize across time because of their recurrent connections. Computations at time step t depend on the previous time state t-1. This inherently sequential nature limits opportunities for parallelism.

Q5)
Formulas:
Conv output size =
((input size + 2padding - filter size - (filter size -1) *(dilation - 1)) / stride) + 1
Max-pooling = ((input size - pooling size) / stride) + 1
Conv parameter = Filter size * Filter size * Input depth + 1
Padding same = 1
Padding valid = 0

layer	Output shape	# of params
1	256*256*64	3*3*3+1
2	124*124*32	5*5*64 +1
3	62*62*32	0
4	62*62*128	3*3*32 + 1
5	23*23*64	5*5*128 + 1
6	11*11*64	0
7	11*11*256	3*3*64 + 1
8	Negative value = -10	5*5*256 + 1
9	Negative value	0

b)
Output = input size ⇒
((input size + 2padding - filter size - (filter size -1) *(dilation - 1)) / stride) + 1 = input size
Padding = (Input size -1)*stride - Input size + filter size + (filters size - 1)(dilation - 1)) /2

Padding = $\frac{(Input\ size - 1) \times Stride - Input\ size + Filter\ size + (Filter\ size - 1) \times (Dilation - 1)}{2}$

Q6)

a)

First sentence: False, batch normalization can increase the speed of all data not just a batch because normalization applied on all batches.

Second sentence: True

Third sentence: False, It just tries to normalize the data, not reducing to zero.

b) Code is available in the directory

c)
The epsilon hyperparameter in batch normalization is a small value, typically around 0.001, that gets added to the estimated variance when normalizing activations. Its purpose is to prevent numeric instability. Because batch norm calculates variance from minibatch samples, those estimates can be inaccurate when batches are small, especially early in training. This could lead to very small or zero variances, which could cause the normalization to explode due to dividing by tiny or zero numbers. Epsilon avoids this by nudging the variance to be a small positive value, ensuring stable normalization without otherwise significantly affecting the overall result. In short, epsilon improves numerical stability when normalizing with batch statistics in situations where the variance may otherwise become tiny.

d)

Using batch normalization with a batch size of 1 causes a few key problems:

1. Unstable statistics: Batch norm calculates the mean and variance across the batch to normalize. With a batch size of 1, the mean and variance are calculated from a single example. This provides extremely noisy and unstable statistics.
2. Potential model failure: Those unstable statistics can easily cause the model to diverge and fail to learn anything. Normalizing by the wrong mean and variance destroys useful signals in the activations.
3. Lack of regularization effects: Some benefits of batch norm come from the regularizing effect of using batch statistics. With batch size 1 you lose this, negating some of the optimizations batch norm is meant to provide.
4. Parameter efficiency loss: Because of the problems above, you lose many benefits batch norm provides in terms of model parameter efficiency and the ability to use higher learning rates without divergence.

In essence, a batch size of 1 removes almost all the desirable properties and benefits of batch normalization. The batch statistics become useless, causing potential model failure. Other batch norm effects like regularization are also removed. This ultimately hurts model training speed, stability, and parameter efficiency.

e)

In a fully connected network with an input size of 10 and output size of 20, the number of trainable parameters without batch normalization would be the product of the input and output sizes (i.e., the weights) plus the output size (i.e., the biases).

So, without batch normalization, the total number of trainable parameters would be:

$10 \times 20 + 20 = 220$

When we add batch normalization, we introduce two additional parameters for each output neuron: `gamma` and `beta`. These are learned parameters that are used to scale and shift the normalized output, respectively.

So, with batch normalization, the total number of trainable parameters would be:

$10 \times 20 + 20 + 2 \times 20 = 260$

Therefore, the fully connected network with batch normalization has **260 trainable parameters**.

Q7)

All the codes are available in the directory

The heatmap shows that the model in the last conv layer focuses on the specific areas that are unique in each class label. Not edges or …
It exports the most unique features for each label.