1 Question 1

Choosing the correct loss function for logistic regression is pivotal. Cross Entropy (CE) is the preferred choice over Mean Squared Error (MSE) for several reasons. Firstly, CE aligns with the probabilistic nature of logistic regression outputs, efficiently measuring the difference between the predicted probabilities and the actual labels. This alignment is crucial because logistic regression models the probability of outcomes, making CE a natural fit for evaluating the accuracy of these probabilistic predictions.

The significance of selecting CE as the loss function extends to its impact on the model's training process. CE loss provides direct feedback on the correctness of the probability estimates, encouraging the model to adjust its weights more effectively to minimize mispredictions. This leads to more stable and faster convergence during training, as the model is guided more efficiently towards the optimal set of weights. In contrast, using MSE can slow down convergence and might not effectively capture the performance of a classification model since it treats the problem as a regression task, not considering the binary or categorical nature of the output.

2 Question 2

For deep neural networks (DNNs) tasked with binary classification, neither Cross Entropy (CE) nor Mean Squared Error (MSE) ensures a convex optimization landscape. The core reason is the inherent complexity and depth of DNNs, which introduce non-linearities, even if the activation functions are linear. DNNs, by design, aim to capture complex patterns through layers of transformations, which results in a non-convex optimization problem regardless of the loss function used.

This complexity means that the optimization landscape of a DNN is filled with numerous local minima, saddle points, and potentially a global minimum that is difficult to identify directly. While convexity guarantees that any local minimum is a global minimum, this property does not hold in the training of DNNs, where the focus shifts towards finding sufficiently good local minima using advanced optimization techniques like stochastic gradient descent (SGD),etc. Thus, selecting CE or MSE as the loss function is more about aligning with the output's nature and the training efficiency rather than ensuring convexity in the optimization problem.

3 Question 3

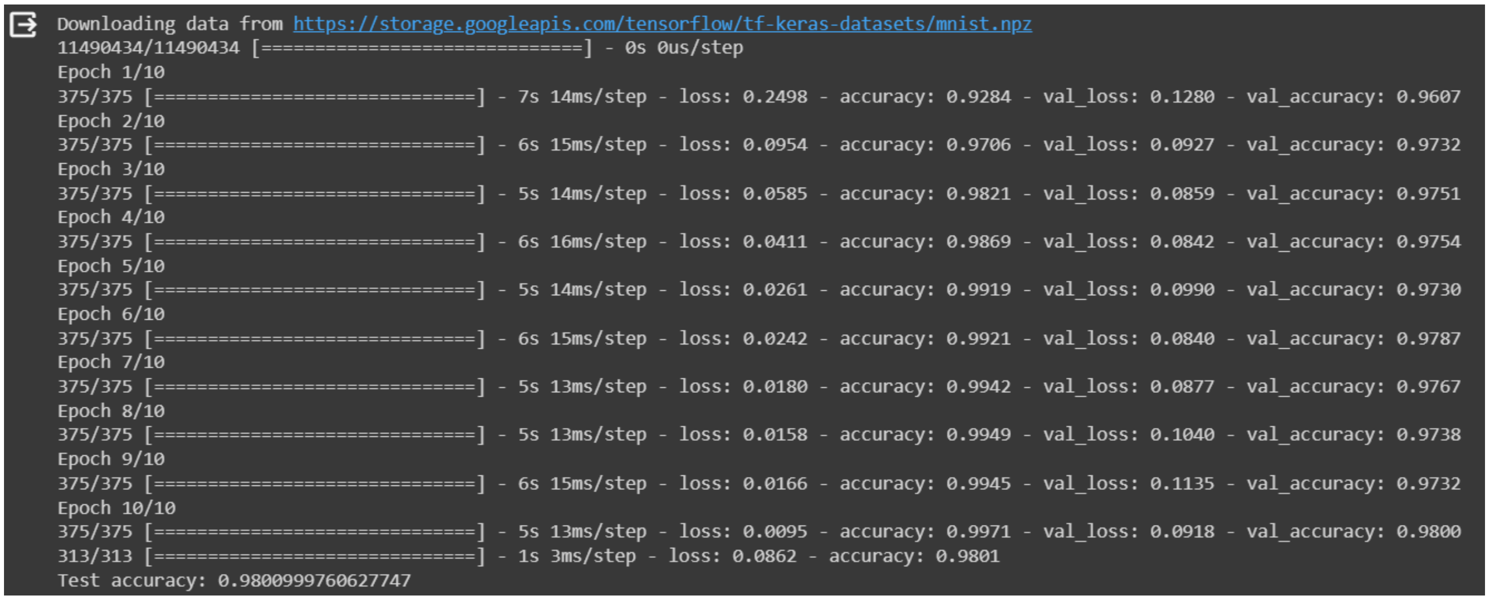
For implementing a dense neural network for a classification task, the network should include an input layer sized according to the dimensionality of the data, several hidden layers with a decreasing number of neurons, and an output layer with a number of neurons corresponding to the number of classes. Each hidden layer should use an activation function to introduce non-linearity, allowing the network to learn complex patterns.

3.1 Preprocessing Input Images

Normalization: Scale the pixel values of images to a range of 0 to 1. This helps in speeding up the convergence during training.
Reshape/Flatten: Since we're using dense layers, input images must be flattened into a 1D array. For example, a 28x28 image would be reshaped to a vector of size 784.

3.2 Hyperparameter Tuning

Learning Rate: Use a learning rate scheduler or experiment with different rates to find an optimal value that ensures convergence without overshooting.
Batch Size: Start with a small batch size and increase it to find a balance between training speed and resource usage.
Epochs: Set a reasonable number of epochs to avoid underfitting or overfitting. Use early stopping to terminate training when the validation loss stops decreasing.
Regularization: Apply techniques like dropout or L2 regularization to prevent overfitting.



```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/10
375/375 [==============================] - 7s 14ms/step - loss: 0.2498 - accuracy: 0.9284 - val_loss: 0.1280 - val_accuracy: 0.9607
Epoch 2/10
375/375 [==============================] - 6s 15ms/step - loss: 0.0954 - accuracy: 0.9706 - val_loss: 0.0927 - val_accuracy: 0.9732
Epoch 3/10
375/375 [==============================] - 5s 14ms/step - loss: 0.0585 - accuracy: 0.9821 - val_loss: 0.0859 - val_accuracy: 0.9751
Epoch 4/10
375/375 [==============================] - 6s 16ms/step - loss: 0.0411 - accuracy: 0.9869 - val_loss: 0.0842 - val_accuracy: 0.9754
Epoch 5/10
375/375 [==============================] - 5s 14ms/step - loss: 0.0261 - accuracy: 0.9919 - val_loss: 0.0990 - val_accuracy: 0.9730
Epoch 6/10
375/375 [==============================] - 6s 15ms/step - loss: 0.0242 - accuracy: 0.9921 - val_loss: 0.0840 - val_accuracy: 0.9787
Epoch 7/10
375/375 [==============================] - 5s 13ms/step - loss: 0.0180 - accuracy: 0.9942 - val_loss: 0.0877 - val_accuracy: 0.9767
Epoch 8/10
375/375 [==============================] - 5s 13ms/step - loss: 0.0158 - accuracy: 0.9949 - val_loss: 0.1040 - val_accuracy: 0.9738
Epoch 9/10
375/375 [==============================] - 6s 15ms/step - loss: 0.0166 - accuracy: 0.9945 - val_loss: 0.1135 - val_accuracy: 0.9732
Epoch 10/10
375/375 [==============================] - 5s 13ms/step - loss: 0.0095 - accuracy: 0.9971 - val_loss: 0.0918 - val_accuracy: 0.9800
313/313 [==============================] - 1s 3ms/step - loss: 0.0862 - accuracy: 0.9801
Test accuracy: 0.9800999760627747
```

Figure 1: Test Accuracy on MNIST Dataset

3.3 Code

```python
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist
from tensorflow.keras.preprocessing.image import ImageDataGenerator

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28 * 28)).astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28)).astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model = models.Sequential([
    layers.Dense(512, activation='relu', input_shape=(28 * 28,)),
    layers.Dense(256, activation='relu'),
    layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10, batch_size=128, validation_split=0.2)

test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')
```

Here we are taking 2 hidden layers and applying this Neural Network on MNIST Dataset.The above code the neural network includes:

Number of Hidden Layers: 2

Neurons per Layer:

The first hidden layer contains 512 neurons The second hidden layer contains 256 neurons.

Activation Functions:

ReLU (Rectified Linear Unit) is used for both hidden layers to introduce non-linearity The output layer uses a Softmax activation function to map the final output to a probability making it suitable for multiclass classification tasks.
QUESTION 3.ipynb

4 Question 4

Among the options LeNet-5, AlexNet, VGG, ResNet, ResNet is often a popular choice due to its ability to learn from a large number of layers without the vanishing gradient problem, owing to residual connections. These properties make ResNet especially suitable for image recognition tasks like classifying the SVHN dataset.

Pretrained models need adaptation for any specific task:

Input Resizing: For ResNet,Alexnet,VGG images typically need to be resized to 224x224 pixels ,so resizing is important. For LeNet the input size is 32x32

Output Layer Modification: We have to relace the final layer to have 10 outputs corresponding to the 10 classes of SVHN digits.

Normalization: Applying the same mean and standard deviation normalization used during pretraining for ResNet, Alexnet and VGG.

4.1 Code

The code to dataset loading, training and evaluating the models are given in the below attached ipynb file.
QUESTION4.ipynb

4.2 Results

Clearly we can see that ResNet performs much better than any of the other models on the SVHN dataset and specifically ResNet-18 is the winner.

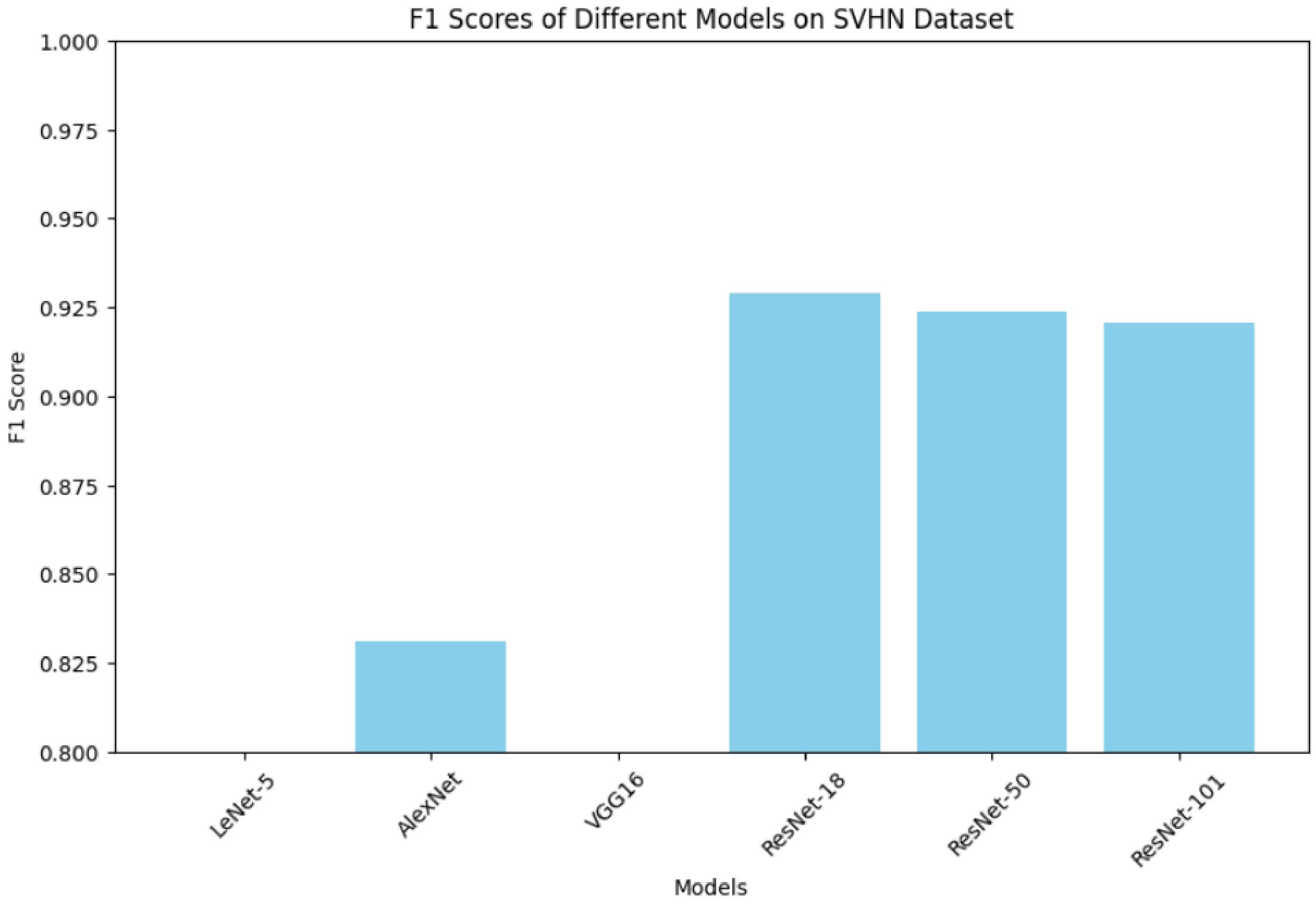


Figure 2: Comparing F1 scores