BUDT737 Final Project Report

Team 29: Phuong Huynh, Saswati Mohanty, Vishal Vindyala, Wanchi Lee

I. Account names and the team name on the Kaggle.com:

Account Name	Score	Method
Phuong Huynh	0.96910	Pytorch
tleeebowl	0.97975	Keras
smohant2	0.99128	Keras

II. Methods used

Our team tried both Pytorch and Keras. Based on the best outcome as shown in the above table, we selected Keras to conduct our final submission.
This is the model training for the best model(smohant) in Keras:

Training Neural network model in Keras

```
[9]  # Using multiple layers of keras layer functions for modelling
     model = keras.Sequential(layers=[
         keras.layers.Conv2D(64,kernel_size=3,input_shape=(28,28,1),activation='relu'),
         keras.layers.AveragePooling2D(pool_size=(2,2)),
         keras.layers.BatchNormalization(),
         keras.layers.GaussianDropout(0.5),
         keras.layers.Conv2D(32,kernel_size=3,input_shape=(28,28,1),activation='relu'),
         keras.layers.AveragePooling2D(pool_size=(2,2)),
         keras.layers.BatchNormalization(),
         keras.layers.GaussianDropout(0.5),
         keras.layers.Flatten(),
         keras.layers.Dense(128,activation='relu'),
         keras.layers.Dense(128,activation='relu'),
         keras.layers.Dense(10, activation="softmax"),
     ])
     #defining the epoch and batch size to fit the model
     model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
     history = model.fit(train_images, train_labels, batch_size=60, epochs=40, validation_split=0.2)
```

The following are the functions used in the keras layers with epoch=40 and batch size=60:

Layer1: Keras Conv2D

We have used this layer twice and it creates a convolution kernel that is convolved with the layer input to produce a tensor of output (where kernel being the convolution matrix thats used for blurring or sharpening etc of the images in image processing. Our respective dimensionalities for the output space are 64,32 with a sernel size of 3 and with activation function relu().

Layer2: Average.Pooling 2d()

We have used this function twice topply average pooling operation for spatial data to achieve a output of tensor with shape: [batchSize, channels, poolsRows, pooledCols] as (pool (2,2)).

Layer3: BatchNormalization()
Batch normalization is also used twice in our model to make a transformation that maintains the mean output close to 0 and the output standard deviation close to 1 which will perform differently for train and inference data.

Layer4: GaussianDropout(0.5)
It is a regularization layer and Gaussian dropout rate for both our layers of this function is 0.5.

Layer5: Flatten()
This flatten layer is the third last layer in our model which flattens the input whilst not affecting the batch size. For inputs without feature axis, flattening adds an extra channel dimension to the output shape.

Layer6: Dense function()

If the input to the layer has a rank greater than 2, then Dense computes the dot product between the inputs and the kernel along the last axis of the inputs and axis 0 of the kernel (using tf.tensordot).

Here we have used three activation functions:

Relu()- Twice :The Rectified Linear Unit is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x it returns that value back.

Softmax()-Once: Dense(activation=softmax) then it will internally create a dense layer first and apply softmax on top it and show you the probability of occurrence.

III. Results and reports

- **Keras**

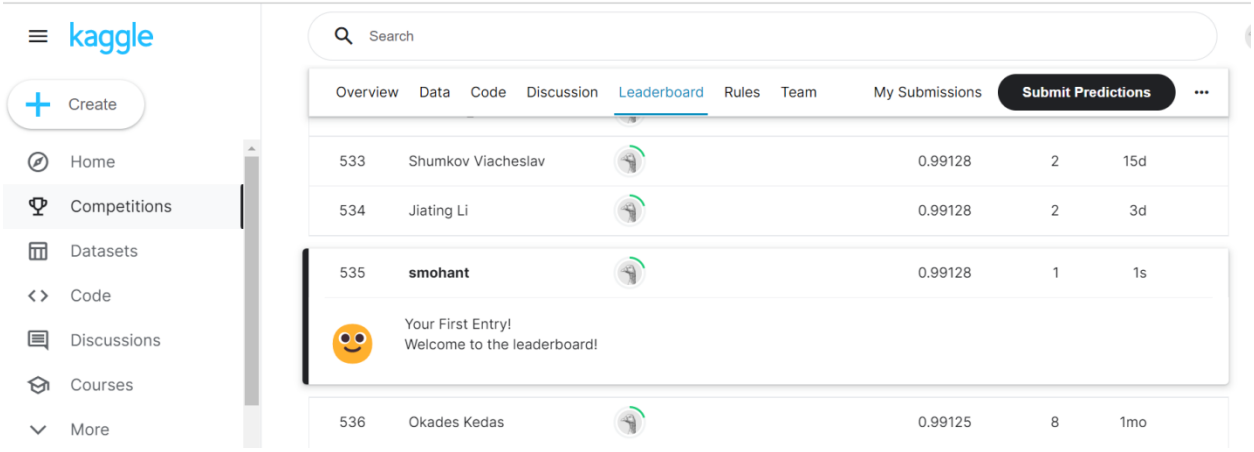


Figure: Kaggle submission of our best model

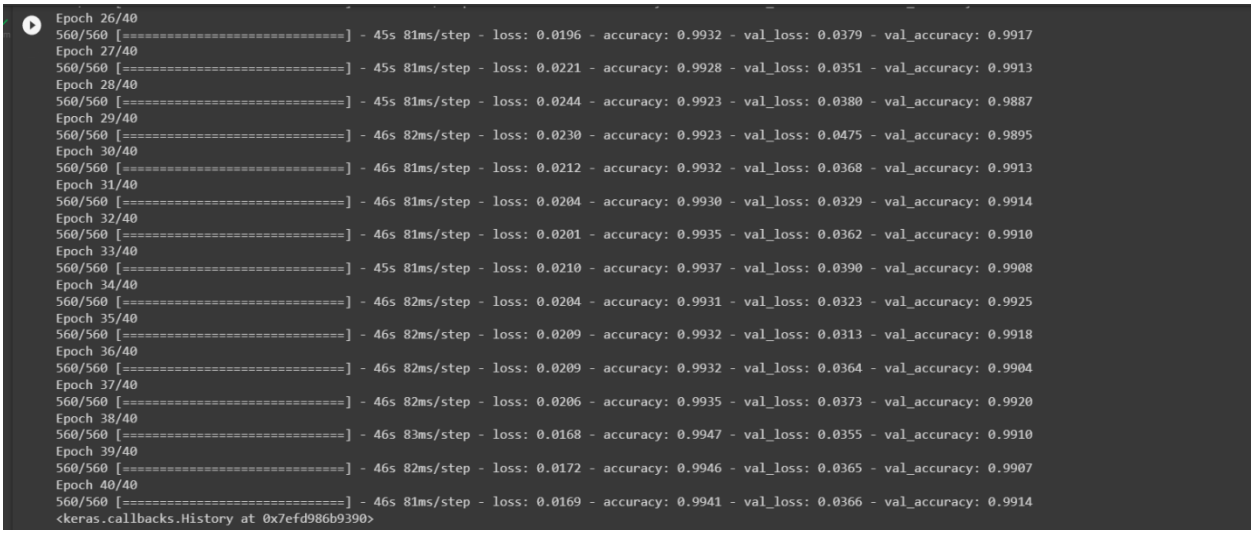


FIgure: Screenshot of our best model(smohant) output in Keras

Here we used an epoch of 40 and a batch size of 60 to achieve an accuracy of 99.128%.

Besides, we would like to provide you some observations not included in our final codes but interesting to find during the model training process:

1. Under keras, we observed that in some cases there was no significant difference in performance between the activation softmax and sigmoid (keeping everything else constant).

Softmax
Epoch 10/10 84/84 [==============================] - 24s 289ms/step - loss: 0.0923 - accuracy: 0.9711
Sigmoid

```
Epoch 10/10
84/84 [==============================] - 24s 284ms/step - loss: 0.0681 - accuracy: 0.9783
```

2. The performance improves along with the increase of the total number of epochs.

Epoch = 20
Epoch 20/20 84/84 [==============================] - 24s 288ms/step - loss: 0.0460 - accuracy: 0.9856
Epoch = 10
Epoch 10/10 84/84 [==============================] - 24s 289ms/step - loss: 0.0923 - accuracy: 0.9711

- **Pytorch**

For PyTorch, although the accuracy is lower than Keras and we did not pick the model to submit, there are also some information and observations that we would like to mention.

We used the min-max normalization to normalize instead of using mean and std.

```
#convert to tensor
#train
train_images_tensor = torch.tensor(train_images)/255.0 #normalization (mean, median)
train_labels_tensor = torch.tensor(train_labels)
train_tensor = TensorDataset(train_images_tensor, train_labels_tensor)

#val
val_images_tensor = torch.tensor(val_images)/255.0
val_labels_tensor = torch.tensor(val_labels)
val_tensor = TensorDataset(val_images_tensor, val_labels_tensor)

#test
test_images_tensor = torch.tensor(test_images)/255.0
```

Then, we applied the rectified linear unit function element-wise, and added another layer as well.

Below is the loss and accuracy for the training data.

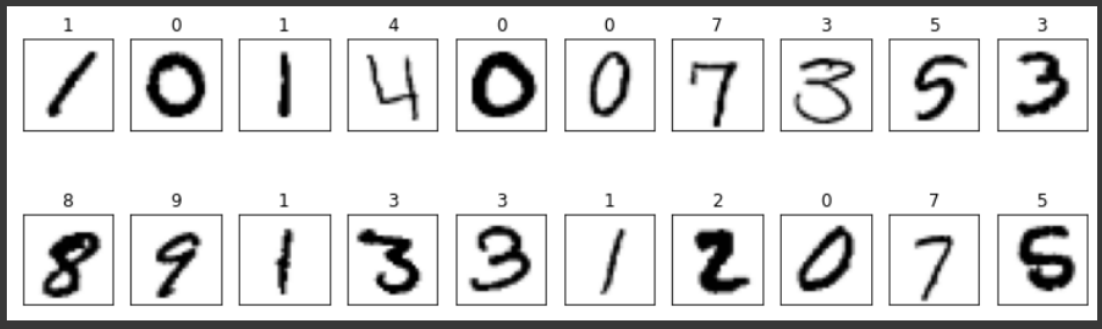
```
Sequential(
  (0): Linear(in_features=784, out_features=256, bias=True)
  (1): ReLU()
  (2): ReLU()
  (3): Linear(in_features=256, out_features=10, bias=True)
  (4): Softmax(dim=1)
)
Epoch 1/25 - loss: 0.07834499620610759 - accuracy: 0.9370833039283752
Epoch 2/25 - loss: 0.07604087717476345 - accuracy: 0.9472321271896362
Epoch 3/25 - loss: 0.07542386535732519 - accuracy: 0.9549999833106995
Epoch 4/25 - loss: 0.07512497915043717 - accuracy: 0.9642857313156128
Epoch 5/25 - loss: 0.07491519456463201 - accuracy: 0.9472321271896362
Epoch 6/25 - loss: 0.07482919892030103 - accuracy: 0.9706249833106995
Epoch 7/25 - loss: 0.07473900014800684 - accuracy: 0.96895831823349
Epoch 8/25 - loss: 0.07464944918240819 - accuracy: 0.9748214483261108
Epoch 9/25 - loss: 0.07461262044097695 - accuracy: 0.9704464077949524
Epoch 10/25 - loss: 0.07456521715791453 - accuracy: 0.975178599357605
Epoch 11/25 - loss: 0.07452125452458859 - accuracy: 0.9660416841506958
Epoch 12/25 - loss: 0.07444396667182446 - accuracy: 0.9779166579246521
Epoch 13/25 - loss: 0.07436562755987758 - accuracy: 0.9739881157875061
Epoch 14/25 - loss: 0.07431924483605794 - accuracy: 0.9662797451019287
Epoch 15/25 - loss: 0.07444272528092066 - accuracy: 0.9736607074737549
Epoch 16/25 - loss: 0.07435729107686452 - accuracy: 0.9737797379493713
Epoch 17/25 - loss: 0.07429565590052378 - accuracy: 0.9772916436195374
Epoch 18/25 - loss: 0.0743862929656392 - accuracy: 0.9745833277702332
Epoch 19/25 - loss: 0.07418265080522923 - accuracy: 0.9790773987770081
Epoch 20/25 - loss: 0.07417031565592402 - accuracy: 0.9803273677825928
Epoch 21/25 - loss: 0.074181229823402 - accuracy: 0.9766368865966797
Epoch 22/25 - loss: 0.07409099782151836 - accuracy: 0.976934552192688
Epoch 23/25 - loss: 0.07418139021666277 - accuracy: 0.979494035243988
Epoch 24/25 - loss: 0.0740873207009974 - accuracy: 0.9815178513526917
Epoch 25/25 - loss: 0.07407115398418336 - accuracy: 0.9705356955528259
```

IV. Data Visualization

1. The training data visualization for 2 rows in Keras best model(smohant):

```
# plot first n images
i = int(input("Enter the number of rows of images: "))
plot(train_images[:i*10], train_labels[:i*10])
```

```
2
```

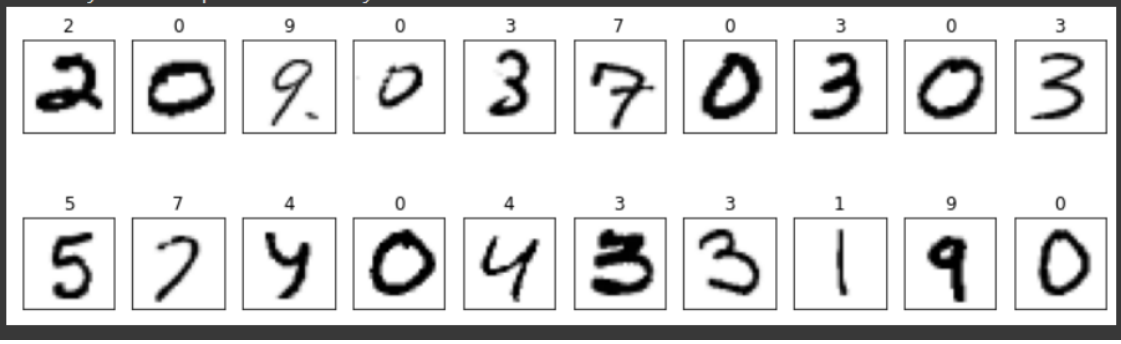


2. Test data visualization for 2 rows in Keras best model(smohant):

Visualizing predictions

```
#Visualizing rows of predicted data
i = int(input("How many rows of prediction do you want to see? "))
plot(test_images[:i*10],preds[:i*10])
```

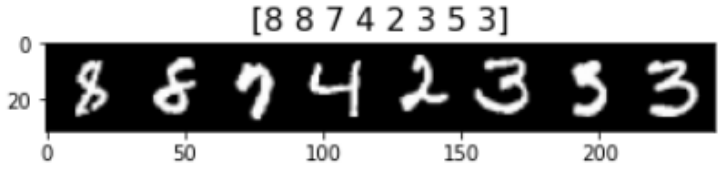
```
How many rows of prediction do you want to see? 2
```



3. Training data and test data visualization for PyTorch model:

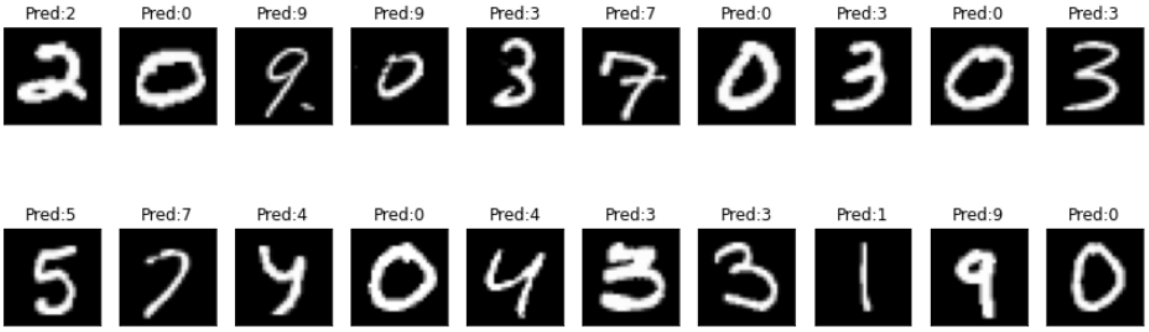
```
for batch_idx, (data, target) in enumerate(train_loader):
    img_grid = make_grid(data[0:8,].unsqueeze(1), nrow=8)
    img_target_labels = target[0:8,].numpy()
    break

plt.imshow(img_grid.numpy().transpose((1,2,0)))
plt.rcParams['figure.figsize'] = (10, 2)
plt.title(img_target_labels, size=16)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/text.py:1165:
  if s != self._text:
```

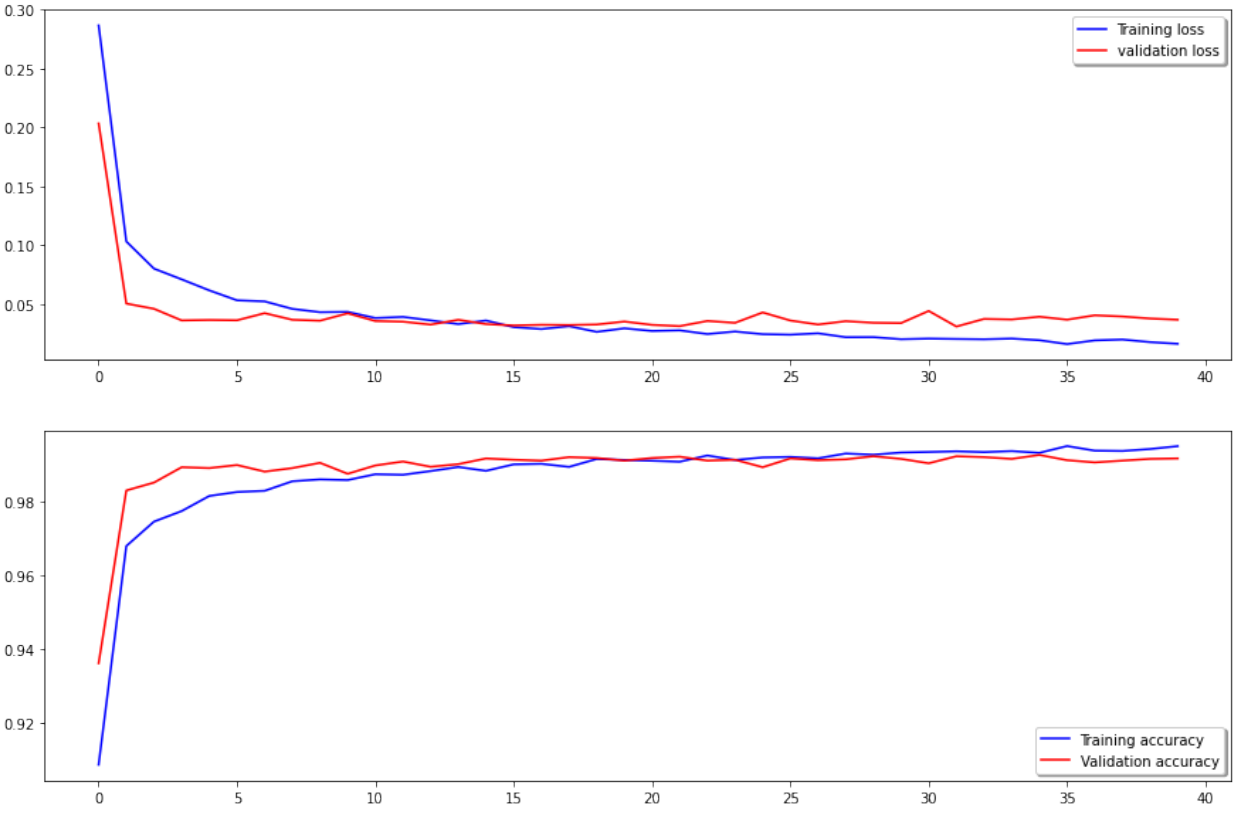


```
fig = plt.figure(figsize=[15, 5])


for i in range(20):
    img = test_images[i].squeeze();
    ax = fig.add_subplot(2, 10, i+1)
    ax.grid(False)
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    ax.title.set_text(f'Pred:{test_preds[i]}')
    plt.imshow(img, cmap='gray')

plt.show()
```

```
<Figure size 1080x360 with 0 Axes>
```

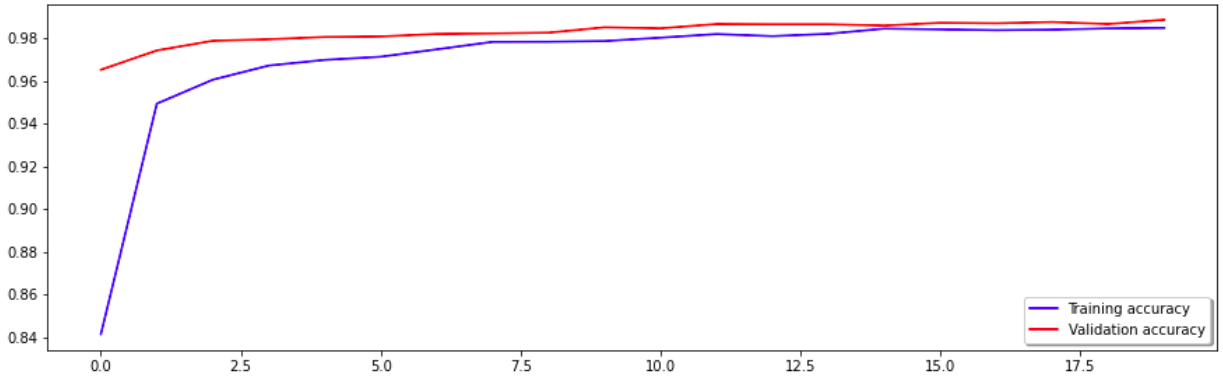


4. For our best model(smohant) the following are the data visualizations for the loss and accuracy functions comparing the training and test models.

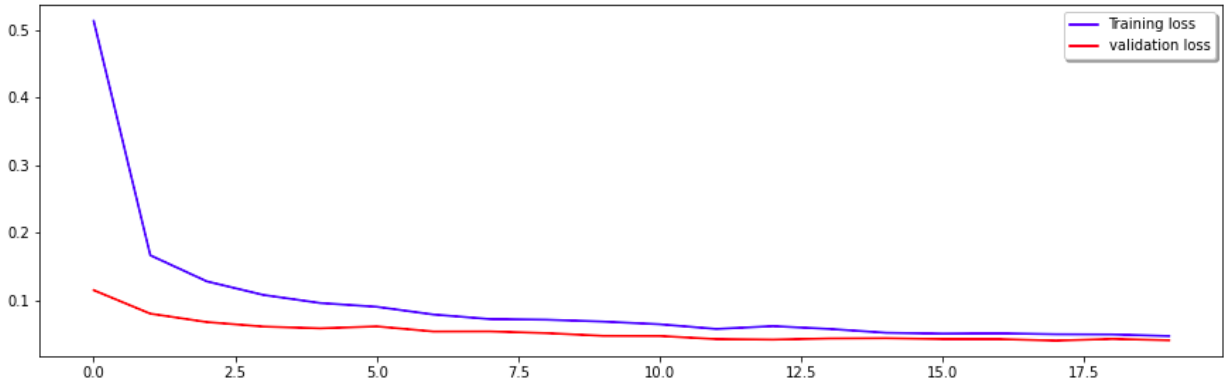


5. The chart displays the accuracy rate and loss rate under Keras method.

Accuracy Rate



Loss



IV. What did you do and what have you learned

1. It was a hands-on experience learning framework building machining learning from scratch. Compared to Pytorch, I feel Keras is more user friendly. For example, we can generate labels from 0-9 simply by to_categorical function. The main steps under Keras approach include: view and glance the data, separate images and labels under training data, reshape and normalize, train models, pick the best based on accuracy rate and loss, and lastly applied to test data.

2. In keras, we learnt about the different combinations of layers and learnt about these functions used. A lot of functions are better to be used at the end of the model as the dense function() as they are known to sum up the probabilities of occurrence at the end of our iteration. The function like flatten() is used once as

we desire to flatten the data as a dimensionally correct output shape to then get into the dense function().

Sources:
The information about the layers of keras are searched from:
https://keras.io/api/layers/normalization_layers/batch_normalization/