Problem Set 2: CNNs [75 pts]

Note: The following has been verified to work with TensorFlow 2.0

* Adapted from official TensorFlow™ tour guide.

TensorFlow is a powerful library for doing large-scale numerical computation. One of the tasks at which it excels is implementing and training deep neural networks. In this assignment you will learn the basic building blocks of a TensorFlow model while constructing a deep convolutional MNIST classifier.

What you are expected to implement in this tutorial:

- Create a softmax regression function that is a model for recognizing MNIST digits, based on looking at every pixel in the image
- Use Tensorflow to train the model to recognize digits by having it "look" at thousands of examples
- Check the model's accuracy with MNIST test data
- Build, train, and test a multilayer convolutional neural network to improve the results

Part 1: Coding [50 pts]

Data

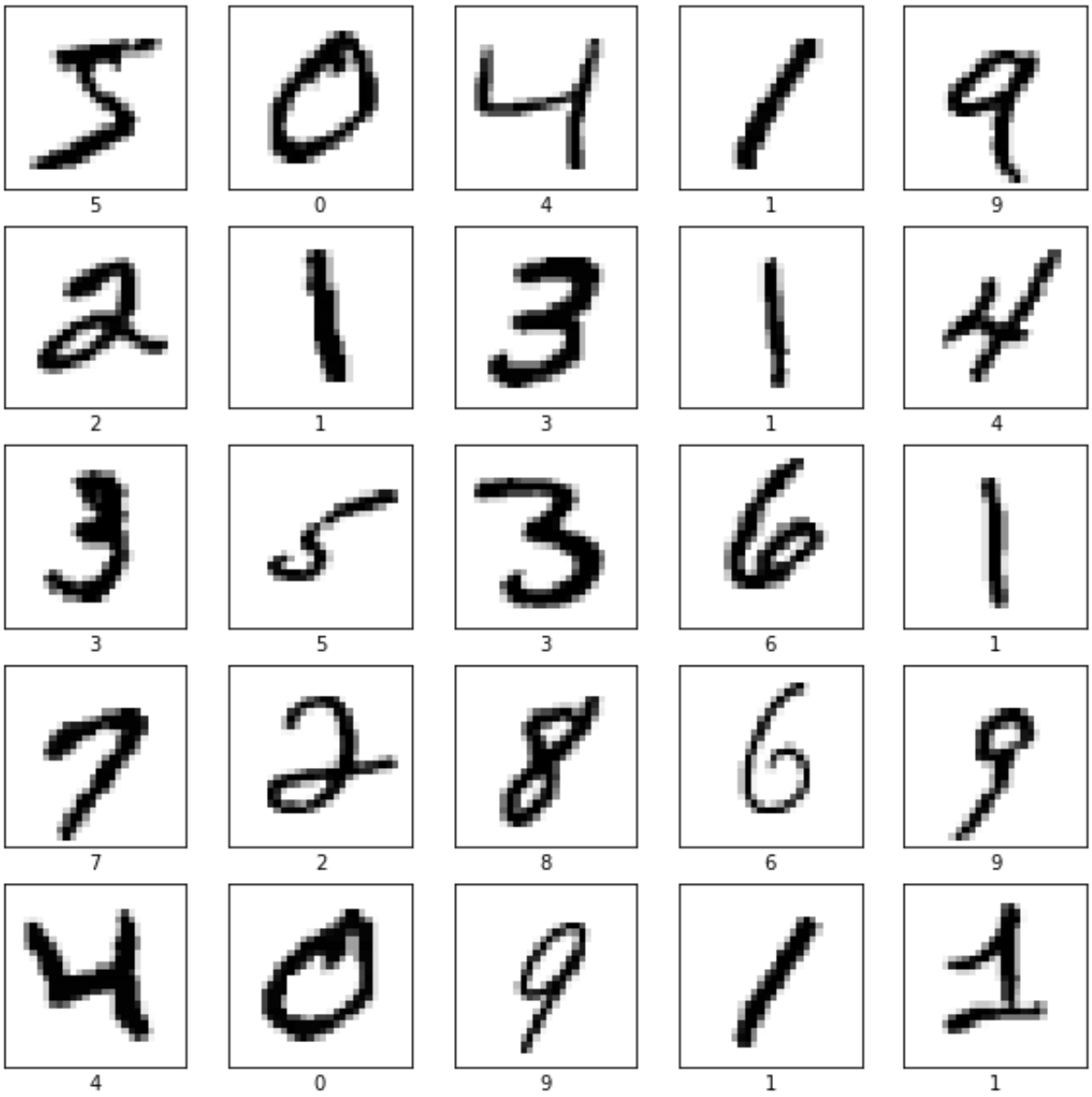
After importing tensorflow, we can download the MNIST dataset with the built-in TensorFlow/Keras method.

In [19]:
```
import os

import tensorflow as tf
import matplotlib.pyplot as plt

os.environ['OMP_NUM_THREADS'] = '1'
tf.__version__
```

Out[19]:
```
'2.0.0'
```

In [20]:

```
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.
class_names = ['0', '1', '2', '3', '4',
               '5', '6', '7', '8', '9']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```



Build the CNN

Build the CNN

In this part we will build a customized TF2 Keras model. As input, a CNN takes tensors of shape (image_height, image_width, color_channels), ignoring the batch size. For MNIST, you will configure our CNN to process inputs of shape (28, 28, 1), which is the format of MNIST images. You can do this by passing the argument input_shape to our first layer.

The overall architecture should be:

```
Model: "customized_cnn"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            multiple                  320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 multiple                  0
_________________________________________________________________
conv2d_3 (Conv2D)            multiple                  18496
_________________________________________________________________
flatten_1 (Flatten)          multiple                  0
_________________________________________________________________
dense_2 (Dense)              multiple                  7930880
_________________________________________________________________
dense_3 (Dense)              multiple                  10250
=================================================================

Total params: 7,959,946
Trainable params: 7,959,946
Non-trainable params: 0
_________________________________________________________________
```

First Convolutional Layer [5 pts]

We can now implement our first layer. The convolution will compute 32 features for each 3x3 patch. The first two dimensions are the patch size, the next is the number of input channels, and the last is the number of output channels.

Max Pooling Layer [5 pts]

We stack max pooling layer after the first convolutional layer. These pooling layers will perform max pooling for each 2x2 patch.

Second Convolutional Layer [5 pts]

In order to build a deep network, we stack several layers of this type. The second layer will have 64 features for each 3x3 patch.

Fully Connected Layers [10 pts]

Now that the image size has been reduced to 11x11, we add a fully-connected layer with 128 neurons to allow processing on the entire image. We reshape the tensor from the second convolutional layer into a batch of vectors before the fully connected layer.

The output layer should also be implemented via a fully connect layer.

Complete the Computation Graph [15 pts]

Please complete the following function:

```
def call(self, inputs, training=None, mask=None):
```

To apply the layer, we first reshape the input to a 4d tensor, with the second and third dimensions corresponding to image width and height, and the final dimension corresponding to the number of color channels (which is 1).

We then convolve the reshaped input with the first convolutional layer and then the max pooling followed by the second convolutional layer. These convolutional layers and the pooling layer will reduce the image size to 11x11.

Dropout Layer [5 pts]

Please add dropouts during training before each fully connected layers, as this helps avoid overfitting during training. https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

In [21]:
```
class CustomizedCNN(tf.keras.models.Model):

    def __init__(self, *args, **kwargs):
        # raise NotImplementedError('Implement Using Keras Layers.')
        super().__init__(*args, **kwargs)

        self.conv2d_2 = tf.keras.layers.Conv2D(filters = 32, kernel_size = (3
                                               input_shape = (28, 28, 1))
        self.max_pooling2d_1 = tf.keras.layers.MaxPooling2D(pool_size = (2, 2
        self.conv2d_3 = tf.keras.layers.Conv2D(filters = 64, kernel_size = (3
        self.flatten_1 = tf.keras.layers.Flatten()
        self.dense_2 = tf.keras.layers.Dense(units = 128, activation = 'relu'
        self.dense_3 = tf.keras.layers.Dense(units = 10)

    def call(self, inputs, training=None, mask=None):
        # raise NotImplementedError('Build the CNN here.')
        inputs = tf.cast(x = tf.expand_dims(inputs, axis = -1), dtype = tf.fl

        conv2d_2 = self.conv2d_2(inputs)
        max_pooling2d_1 = self.max_pooling2d_1(conv2d_2)
        conv2d_3 = self.conv2d_3(max_pooling2d_1)
        flatten_1 = self.flatten_1(conv2d_3)

        if training:
            flatten_1 = tf.nn.dropout(flatten_1, 0.5)
        dense_2 = self.dense_2(flatten_1)
        if training:
            dense_2 = tf.nn.dropout(dense_2, 0.5)
        dense_3 = self.dense_3(dense_2)

        return dense_3
```

Build the Model

In [22]:
```
model = CustomizedCNN()
model.build(input_shape=(None, 28, 28))
model.summary()
```

```
Model: "customized_cnn_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            multiple                  320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 multiple                  0
_________________________________________________________________
conv2d_9 (Conv2D)            multiple                  18496
_________________________________________________________________
flatten_3 (Flatten)          multiple                  0
_________________________________________________________________
dense_6 (Dense)              multiple                  991360
_________________________________________________________________
dense_7 (Dense)              multiple                  1290
=================================================================
Total params: 1,011,466
Trainable params: 1,011,466
Non-trainable params: 0
_________________________________________________________________
```

We can specify a loss function just as easily. Loss indicates how bad the model's prediction was on a single example; we try to minimize that while training across all the examples. Here, our loss function is the cross-entropy between the target and the softmax activation function applied to the model's prediction. As in the beginners tutorial, we use the stable formulation:

In [23]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logit
              metrics=['accuracy'])
```

Train and Evaluate the Model [5 pts]

We will use a more sophisticated ADAM optimizer instead of a Gradient Descent Optimizer.

Feel free to run this code. Be aware that it does 10 training epochs and may take a while (possibly up to half an hour), depending on your processor.

The final test set accuracy after running this code should be approximately 98.7% -- not state of the art, but respectable.

We have learned how to quickly and easily build, train, and evaluate a fairly sophisticated deep learning model using TensorFlow.

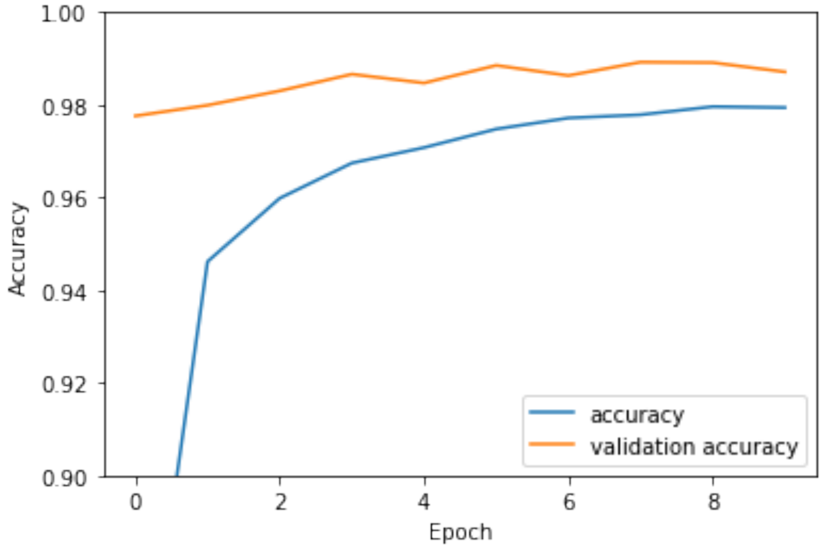
In [24]:
```
# raise NotImplementedError('Update correct arguments for the fit method belo
history = model.fit(train_images, train_labels, epochs = 10, validation_data =
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 84s 1ms/sample - loss: 0.8896 -
accuracy: 0.8373 - val_loss: 0.0740 - val_accuracy: 0.9776
Epoch 2/10
60000/60000 [==============================] - 84s 1ms/sample - loss: 0.1907 -
accuracy: 0.9462 - val_loss: 0.0614 - val_accuracy: 0.9799
Epoch 3/10
60000/60000 [==============================] - 84s 1ms/sample - loss: 0.1396 -
accuracy: 0.9598 - val_loss: 0.0526 - val_accuracy: 0.9830
Epoch 4/10
60000/60000 [==============================] - 83s 1ms/sample - loss: 0.1140 -
accuracy: 0.9674 - val_loss: 0.0422 - val_accuracy: 0.9866
Epoch 5/10
60000/60000 [==============================] - 83s 1ms/sample - loss: 0.1001 -
accuracy: 0.9708 - val_loss: 0.0431 - val_accuracy: 0.9847
Epoch 6/10
60000/60000 [==============================] - 83s 1ms/sample - loss: 0.0867 -
accuracy: 0.9748 - val_loss: 0.0340 - val_accuracy: 0.9885
Epoch 7/10
60000/60000 [==============================] - 83s 1ms/sample - loss: 0.0776 -
accuracy: 0.9772 - val_loss: 0.0451 - val_accuracy: 0.9863
Epoch 8/10
60000/60000 [==============================] - 82s 1ms/sample - loss: 0.0746 -
accuracy: 0.9779 - val_loss: 0.0327 - val_accuracy: 0.9892
Epoch 9/10
60000/60000 [==============================] - 82s 1ms/sample - loss: 0.0697 -
accuracy: 0.9796 - val_loss: 0.0351 - val_accuracy: 0.9891
Epoch 10/10
60000/60000 [==============================] - 82s 1ms/sample - loss: 0.0690 -
accuracy: 0.9794 - val_loss: 0.0437 - val_accuracy: 0.9871
```

In [25]:

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.9, 1])
plt.legend(loc='lower right')
```

Out[25]: <matplotlib.legend.Legend at 0x7ffd95c811d0>



In [26]:

```
test_loss, test_acc = model.evaluate(test_images,  test_labels)
print(test_acc)
```

```
10000/1 [======================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
===============================================================================
```

```
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
====================] - 6s 610us/sample - loss: 0.0220 - accuracy: 0.9871
0.9871
```

Part 2: Written [22 pts]

2.1 [**4pts**] True/False, and why? Training a neural network from scratch takes significantly less time than using a pre-trained network and fine-tuning the final layers on the new task at hand.

Solution Here

false

actually, using a pre-trained network is faster than training a nn from scratch, because a pre-trained model is a model created by some one else to solve a similar problem, and fine-tuning the final layers only need to reconstruct the last layer base on the model

2.2 [**2pts**] Which of the above approaches in 2.1 is referred to as *Transfer Learning*?

Solution Here

the pre-train & fine-tuning

"using a pre-trained network and fine-tuning the final layers on the new task at hand"

2.3 [**4pts**] List two ways to downsize feature maps in convolutional neural networks.

Solution Here

1. stride > 1, for example stride = 2
2. pooling, for example max pooling

2.4 [**4pts**] List four techniques that help a model avoid overfitting briefly explaining each.

Solution Here

1. L2/L1 regularization: restrictions on parameter values or adding terms to the objective function
2. Data Set Augmentation: augment our existing data set, because, usually, the more data, the better for generalization
3. Early Stopping: stop training when generalization error increases
4. Dropout: drop out some data points randomly to prevent complex co-adaptations on training data
5. Minibatch Training: a variation of the gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients

2.5 [2pts] Read the following on convolutions: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1. What is padding? And what does it help us achieve.

Solution Here

pad the edges with extra, “fake” pixels (usually of value 0, hence the oft-used term “zero padding”)

therefore, the kernel when sliding can allow the original edge pixels to be at its center, while extending into the fake pixels beyond the edge, producing an output the same size as the input

2.6 [2pts] How does back-propagation in RNNs differ from back-propagation in CNNs?

Solution Here

in RNNS, we use back-propagation through time:

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

where

$$\frac{\partial E_t}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

but in CNNs, we don't need to deal with timestep:

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial h} \frac{\partial h}{\partial W}$$

2.7 [**4pts**] Explain two different ways to reduce the dimensionality of 1024D feature vectors extracted from a CNN for images.

Solution Here

autoencoder: use the bottleneck layer to reduce the dimensionality

pca: by using pca, we can choose subspace with minimal “information loss” and so that reduce the dimensionality meanwhile keep all the important informantion

2.8 [**3pts**] What is an auto-encoder? What can it be used for?

Solution Here

an autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs