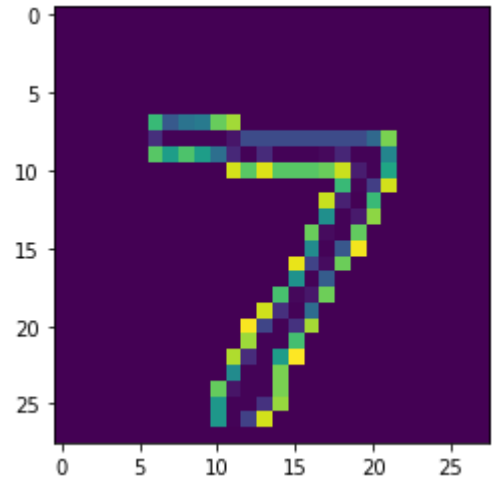
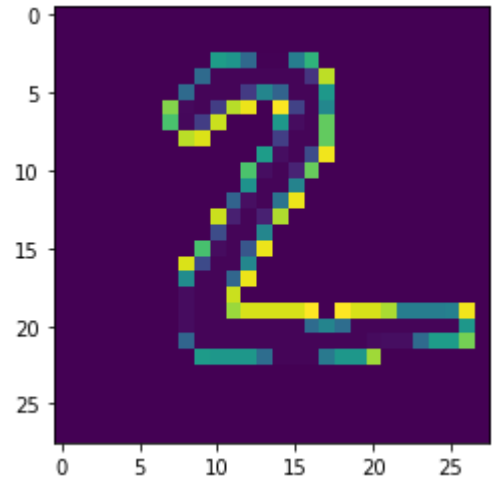
```
#reference https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-k
# reference https://elitedatascience.com/keras-tutorial-deep-learning-in-python

from keras.datasets import mnist
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout
import numpy as np


def gen_image(arr):
    two_d = (np.reshape(arr, (28, 28)) * 255).astype(np.uint8)
    plt.imshow(two_d, interpolation='nearest')
    return plt


#download mnist data and split into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

#check image shape
X_train[0].shape

#normalization values between 0 and 1
#X_train = X_train / 255
#X_test = X_test / 255

#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)

#one-hot encode target column which is equal to generate_t in program 5
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]


#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
        # 64 are the number of filters, kernel size is the size of the filters example 3*3
#model.add(Conv2D(32, kernel_size=3, activation='relu'))
#model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

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

```
                metrics=['accuracy'])

# 9. Fit model on training data
model.fit(X_train, y_train,
          batch_size=32, nb_epoch=10, verbose=1) #epochs  = iterations(Nit)

# 10. Evaluate model on test data
score = model.evaluate(X_test, y_test, verbose=1)

print('Testing accuracy - > ',score[1] * 100)

ytested = model.predict_classes(X_test)
for i in range(4):
  gen_image(X_test[i]).show() # printing image vs the predicted image below
  print("The Predicted Testing image is =%s" % (ytested[i]))
```

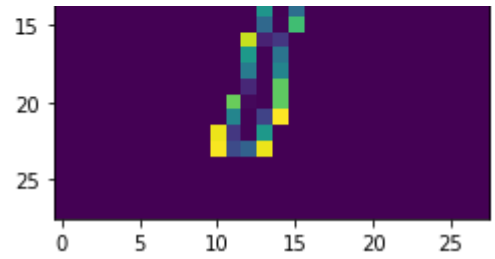
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:56: UserWarning: The `nb
Epoch 1/10
60000/60000 [==============================] - 11s 183us/step - loss: 11.9718 - acc:
Epoch 2/10
60000/60000 [==============================] - 11s 176us/step - loss: 8.7755 - acc: 0
Epoch 3/10
60000/60000 [==============================] - 11s 175us/step - loss: 7.1114 - acc: 0
Epoch 4/10
60000/60000 [==============================] - 11s 178us/step - loss: 5.8809 - acc: 0
Epoch 5/10
60000/60000 [==============================] - 11s 176us/step - loss: 5.6240 - acc: 0
Epoch 6/10
60000/60000 [==============================] - 11s 178us/step - loss: 5.3556 - acc: 0
Epoch 7/10
60000/60000 [==============================] - 11s 183us/step - loss: 5.2619 - acc: 0
Epoch 8/10
60000/60000 [==============================] - 11s 176us/step - loss: 5.2982 - acc: 0
Epoch 9/10
60000/60000 [==============================] - 11s 177us/step - loss: 5.0499 - acc: 0
Epoch 10/10
60000/60000 [==============================] - 11s 177us/step - loss: 5.1202 - acc: 0
10000/10000 [==============================] - 1s 59us/step
Testing accuracy - >  73.09
```



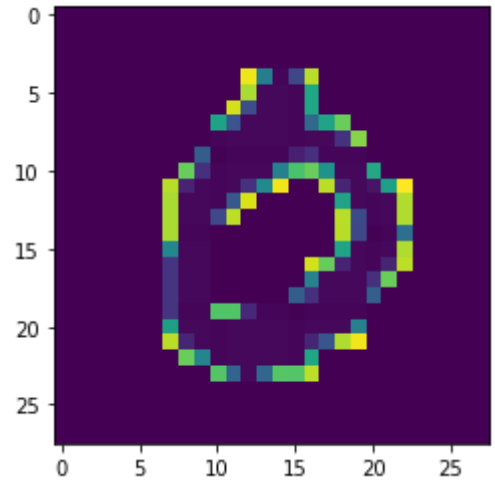

```
The Predicted Testing image is =7
```




```
The Predicted Testing image is =2
```



```
The Predicted Testing image is =1
```



```
The Predicted Testing image is =6
```

```
3+4
```

```
score
```

```
[0.026925411104456, 0.9924]
```

```
score[1] * 100
```

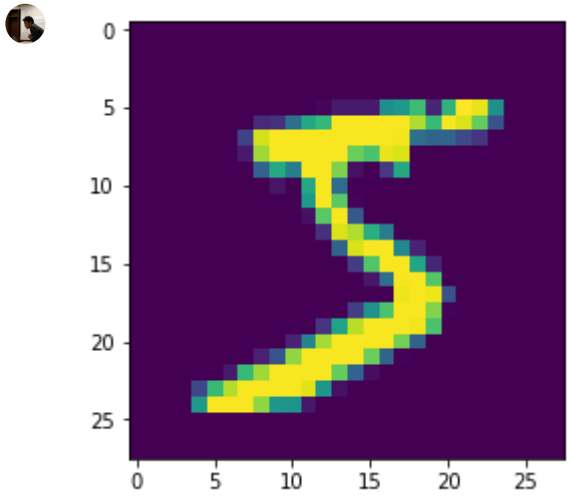
```
99.27
```

```
# 3. Import libraries and modules
import numpy as np
np.random.seed(123)  # for reproducibility

from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.utils import np_utils
from keras.datasets import mnist

# 4. Load pre-shuffled MNIST data into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# 5. Preprocess input data
img_rows = X_train.shape[1]
img_cols = X_train.shape[2]

X_train = X_train.reshape(X_train.shape[0], img_cols, img_rows, 1)
X_test = X_test.reshape(X_test.shape[0], img_cols, img_rows, 1)
#X_train = X_train.reshape(X_train.shape[0], 1, 28, 28)
#X_test = X_test.reshape(X_test.shape[0], 1, 28, 28)
```

```
#X_train = X_train.astype('float32')
#X_test = X_test.astype('float32')
X_train = X_train / 255
X_test = X_test /255

# 6. Preprocess class labels
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)

# 7. Define model architecture
model = Sequential()

model.add(Convolution2D(32, 3, 3, activation='relu', input_shape=(1,28,28)))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

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

# 9. Fit model on training data
model.fit(X_train, Y_train,
          batch_size=32, nb_epoch=10, verbose=1)

# 10. Evaluate model on test data
score = model.evaluate(X_test, Y_test, verbose=0)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:33: UserWarning: Update
---------------------------------------------------------------------------
InvalidArgumentError                      Traceback (most recent call last)
```

```
import matplotlib.pyplot as plt
#plot the first image in the dataset
plt.imshow(X_train[0])
#check image shape
X_train[0].shape

#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
```




```
from keras.utils import to_categorical
#one-hot encode target column
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]

#X_train = X_train.reshape(60000,28,28,1)
#X_test = X_test.reshape(10000,28,28,1)
#X_train = X_train.reshape((-1, 1, 100, 1))
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten
#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
```

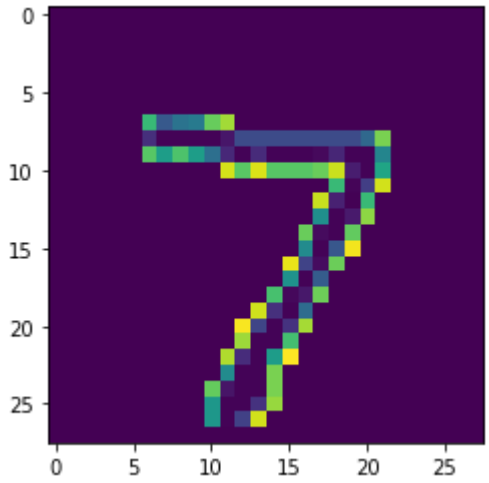
```
model.add(Flatten())
model.add(Dense(10, activation='softmax'))

#compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

#train the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3)

#predict first 4 images in the test set
model.predict(X_test[:4])

#actual results for first 4 images in test set
y_test[:4]
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 25s 423us/step - loss: 9.5803 - acc: 0
Epoch 2/3
60000/60000 [==============================] - 18s 303us/step - loss: 7.7504 - acc: 0
Epoch 3/3
60000/60000 [==============================] - 18s 304us/step - loss: 7.5434 - acc: 0
array([[0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

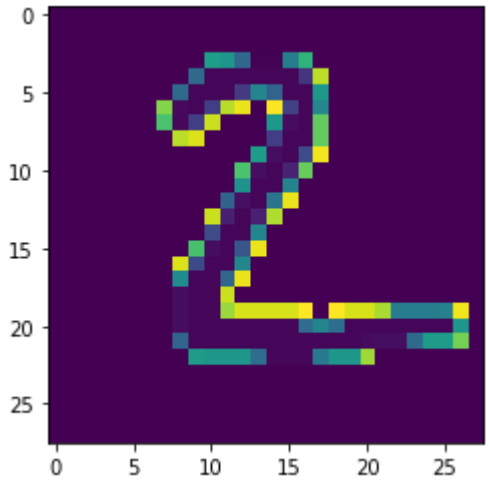
```
#reference https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-k
# reference https://elitedatascience.com/keras-tutorial-deep-learning-in-python

from keras.datasets import mnist
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout
import numpy as np


def gen_image(arr):
    two_d = (np.reshape(arr, (28, 28)) * 255).astype(np.uint8)
    plt.imshow(two_d, interpolation='nearest')
    return plt


#download mnist data and split into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

#check image shape
X_train[0].shape

#normalization values between 0 and 1
#X_train = X_train / 255
#X_test = X_test / 255

#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)

#one-hot encode target column which is equal to generate_t in program 5
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]


#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
        # 64 are the number of filters, kernel size is the size of the filters example 3*3
#model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

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

```
                metrics=['accuracy'])

# 9. Fit model on training data
model.fit(X_train, y_train,
          batch_size=32, nb_epoch=10, verbose=1) #epochs  = iterations(Nit)

# 10. Evaluate model on test data
score = model.evaluate(X_test, y_test, verbose=1)

print('Testing accuracy - > ',score[1] * 100)

ytested = model.predict_classes(X_test)
for i in range(4):
  gen_image(X_test[i]).show() # printing image vs the predicted image below
  print("The Predicted Testing image is =%s" % (ytested[i]))
```

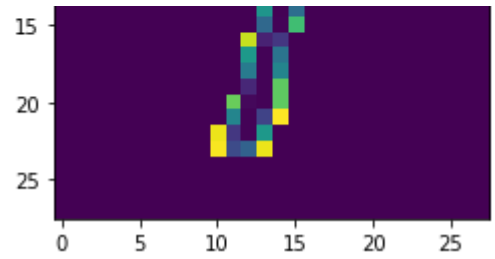
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:56: UserWarning: The `nb
Epoch 1/10
60000/60000 [==============================] - 8s 133us/step - loss: 4.2292 - acc: 0.
Epoch 2/10
60000/60000 [==============================] - 8s 126us/step - loss: 0.1996 - acc: 0.
Epoch 3/10
60000/60000 [==============================] - 7s 124us/step - loss: 0.1527 - acc: 0.
Epoch 4/10
60000/60000 [==============================] - 7s 125us/step - loss: 0.1437 - acc: 0.
Epoch 5/10
60000/60000 [==============================] - 7s 125us/step - loss: 0.1250 - acc: 0.
Epoch 6/10
60000/60000 [==============================] - 8s 125us/step - loss: 0.1193 - acc: 0.
Epoch 7/10
60000/60000 [==============================] - 7s 125us/step - loss: 0.1069 - acc: 0.
Epoch 8/10
60000/60000 [==============================] - 7s 125us/step - loss: 0.1064 - acc: 0.
Epoch 9/10
60000/60000 [==============================] - 7s 125us/step - loss: 0.1027 - acc: 0.
Epoch 10/10
60000/60000 [==============================] - 8s 132us/step - loss: 0.1028 - acc: 0.
10000/10000 [==============================] - 1s 56us/step
Testing accuracy - >  98.21
```



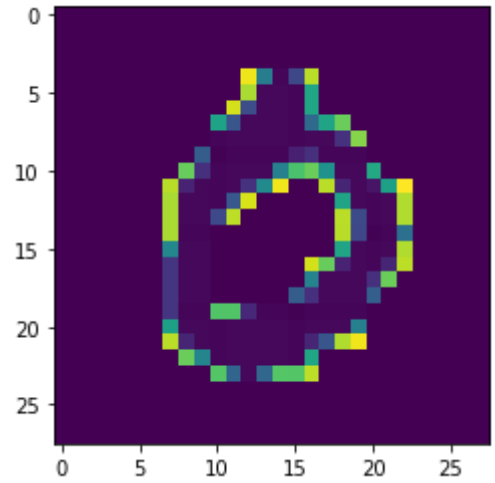

```
The Predicted Testing image is =7
```




```
The Predicted Testing image is =2
```



```
The Predicted Testing image is =1
```



```
The Predicted Testing image is =0
```

```
3+4
```

```
score
```

```
[0.026925411104456, 0.9924]
```

```
score[1] * 100
```

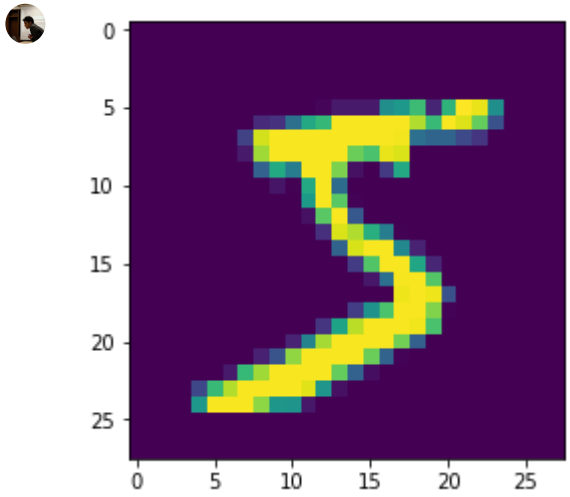
```
99.27
```

```
# 3. Import libraries and modules
import numpy as np
np.random.seed(123)  # for reproducibility

from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.utils import np_utils
from keras.datasets import mnist

# 4. Load pre-shuffled MNIST data into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# 5. Preprocess input data
img_rows = X_train.shape[1]
img_cols = X_train.shape[2]

X_train = X_train.reshape(X_train.shape[0], img_cols, img_rows, 1)
X_test = X_test.reshape(X_test.shape[0], img_cols, img_rows, 1)
#X_train = X_train.reshape(X_train.shape[0], 1, 28, 28)
#X_test = X_test.reshape(X_test.shape[0], 1, 28, 28)
```

```
#X_train = X_train.astype('float32')
#X_test = X_test.astype('float32')
X_train = X_train / 255
X_test = X_test /255

# 6. Preprocess class labels
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)

# 7. Define model architecture
model = Sequential()

model.add(Convolution2D(32, 3, 3, activation='relu', input_shape=(1,28,28)))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

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

# 9. Fit model on training data
model.fit(X_train, Y_train,
          batch_size=32, nb_epoch=10, verbose=1)

# 10. Evaluate model on test data
score = model.evaluate(X_test, Y_test, verbose=0)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:33: UserWarning: Update
---------------------------------------------------------------------------
InvalidArgumentError                      Traceback (most recent call last)
```

```
import matplotlib.pyplot as plt
#plot the first image in the dataset
plt.imshow(X_train[0])
#check image shape
X_train[0].shape

#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
```




```
from keras.utils import to_categorical
#one-hot encode target column
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]

#X_train = X_train.reshape(60000,28,28,1)
#X_test = X_test.reshape(10000,28,28,1)
#X_train = X_train.reshape((-1, 1, 100, 1))
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten
#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
```

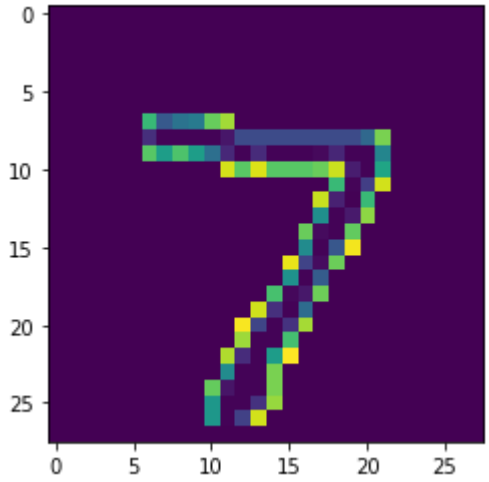
```
model.add(Flatten())
model.add(Dense(10, activation='softmax'))

#compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

#train the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3)

#predict first 4 images in the test set
model.predict(X_test[:4])

#actual results for first 4 images in test set
y_test[:4]
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 25s 423us/step - loss: 9.5803 - acc: 0
Epoch 2/3
60000/60000 [==============================] - 18s 303us/step - loss: 7.7504 - acc: 0
Epoch 3/3
60000/60000 [==============================] - 18s 304us/step - loss: 7.5434 - acc: 0
array([[0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

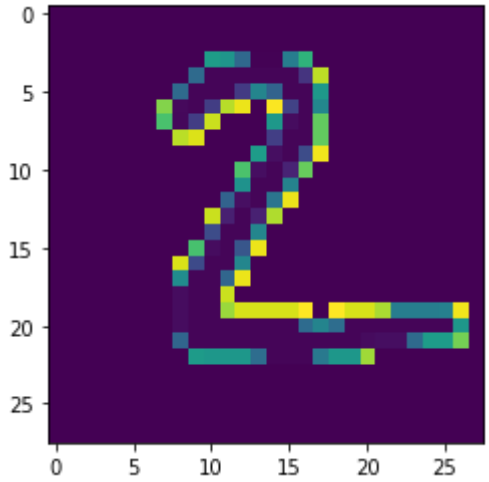
```
#reference https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-k
# reference https://elitedatascience.com/keras-tutorial-deep-learning-in-python

from keras.datasets import mnist
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout
import numpy as np


def gen_image(arr):
    two_d = (np.reshape(arr, (28, 28)) * 255).astype(np.uint8)
    plt.imshow(two_d, interpolation='nearest')
    return plt


#download mnist data and split into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

#check image shape
X_train[0].shape

#normalization values between 0 and 1
#X_train = X_train / 255
#X_test = X_test / 255

#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)

#one-hot encode target column which is equal to generate_t in program 5
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]


#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
        # 64 are the number of filters, kernel size is the size of the filters example 3*3
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

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

```
                metrics=['accuracy'])

# 9. Fit model on training data
model.fit(X_train, y_train,
          batch_size=32, nb_epoch=10, verbose=1) #epochs  = iterations(Nit)

# 10. Evaluate model on test data
score = model.evaluate(X_test, y_test, verbose=1)

print('Testing accuracy - > ',score[1] * 100)

ytested = model.predict_classes(X_test)
for i in range(4):
  gen_image(X_test[i]).show() # printing image vs the predicted image below
  print("The Predicted Testing image is =%s" % (ytested[i]))
```

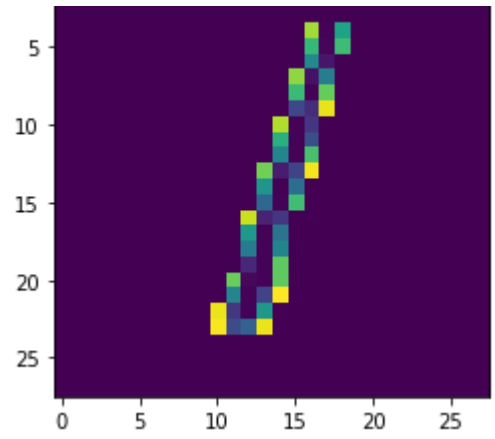
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

60000/60000 [==============================] - 28s 460us/step - loss: 2.9217 - acc: 0
Epoch 2/10
60000/60000 [==============================] - 20s 335us/step - loss: 0.1529 - acc: 0
Epoch 3/10
60000/60000 [==============================] - 20s 331us/step - loss: 0.1185 - acc: 0
Epoch 4/10
60000/60000 [==============================] - 20s 329us/step - loss: 0.1000 - acc: 0
Epoch 5/10
60000/60000 [==============================] - 20s 330us/step - loss: 0.0890 - acc: 0
Epoch 6/10
60000/60000 [==============================] - 20s 329us/step - loss: 0.0820 - acc: 0
Epoch 7/10
60000/60000 [==============================] - 20s 330us/step - loss: 0.0753 - acc: 0
Epoch 8/10
60000/60000 [==============================] - 20s 329us/step - loss: 0.0715 - acc: 0
Epoch 9/10
60000/60000 [==============================] - 20s 328us/step - loss: 0.0681 - acc: 0
Epoch 10/10
60000/60000 [==============================] - 20s 332us/step - loss: 0.0638 - acc: 0
10000/10000 [==============================] - 1s 107us/step
Testing accuracy - >  98.85000000000001
```



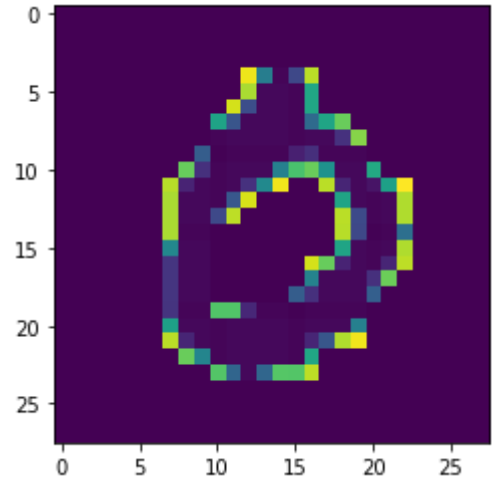

```
The Predicted Testing image is =7
```




```
The Predicted Testing image is =2
```




```
The Predicted Testing image is =1
```




```
The Predicted Testing image is =0
```

```
3+4
```

```
score
```

```
score[1] * 100
```

```
# 3. Import libraries and modules
import numpy as np
np.random.seed(123)  # for reproducibility

from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.utils import np_utils
from keras.datasets import mnist

# 4. Load pre-shuffled MNIST data into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# 5. Preprocess input data
img_rows = X_train.shape[1]
img_cols = X_train.shape[2]

X_train = X_train.reshape(X_train.shape[0], img_cols, img_rows, 1)
X_test = X_test.reshape(X_test.shape[0], img_cols, img_rows, 1)
#X_train = X_train.reshape(X_train.shape[0], 1, 28, 28)
#X_test = X_test.reshape(X_test.shape[0], 1, 28, 28)
#X_train = X_train.astype('float32')
#X_test = X_test.astype('float32')
X_train = X_train / 255
X_test = X_test /255

# 6. Preprocess class labels
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)

# 7. Define model architecture
model = Sequential()

model.add(Convolution2D(32, 3, 3, activation='relu', input_shape=(1,28,28)))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
```

```
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

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

# 9. Fit model on training data
model.fit(X_train, Y_train,
          batch_size=32, nb_epoch=10, verbose=1)

# 10. Evaluate model on test data
score = model.evaluate(X_test, Y_test, verbose=0)
```

```
import matplotlib.pyplot as plt
#plot the first image in the dataset
plt.imshow(X_train[0])
#check image shape
X_train[0].shape

#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
```

```
from keras.utils import to_categorical
#one-hot encode target column
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]

#X_train = X_train.reshape(60000,28,28,1)
#X_test = X_test.reshape(10000,28,28,1)
#X_train = X_train.reshape((-1, 1, 100, 1))
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten
#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))

#compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

#train the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3)

#predict first 4 images in the test set
model.predict(X_test[:4])

#actual results for first 4 images in test set
y_test[:4]
```

```
#reference https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-k
# reference https://elitedatascience.com/keras-tutorial-deep-learning-in-python

from keras.datasets import mnist
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout
import numpy as np


def gen_image(arr):
    two_d = (np.reshape(arr, (28, 28)) * 255).astype(np.uint8)
    plt.imshow(two_d, interpolation='nearest')
    return plt


#download mnist data and split into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

#check image shape
X_train[0].shape

#normalization values between 0 and 1
X_train = X_train / 255
X_test = X_test / 255

#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)

#one-hot encode target column which is equal to generate_t in program 5
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]


#create model
model = Sequential()
#add model layers

model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
        # 64 are the number of filters, kernel size is the size of the filters example 3*3
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

# 8. Compile model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
```

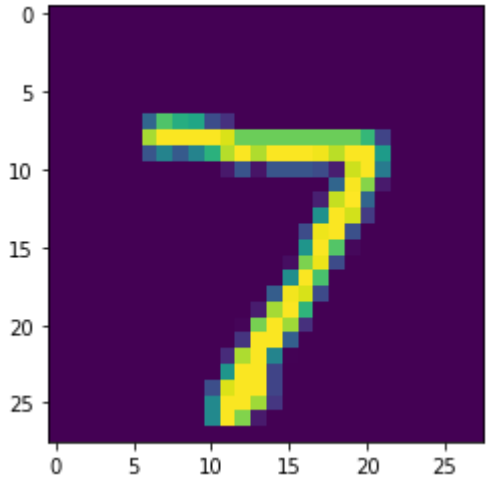
```
                optimizer='adam',
                metrics=['accuracy'])

# 9. Fit model on training data
model.fit(X_train, y_train,
          batch_size=32, nb_epoch=10, verbose=1) #epochs  = iterations(Nit)

# 10. Evaluate model on test data
score = model.evaluate(X_test, y_test, verbose=1)

print('Testing accuracy - > ',score[1] * 100)

ytested = model.predict_classes(X_test)
for i in range(4):
  gen_image(X_test[i]).show() # printing image vs the predicted image below
  print("The Predicted Testing image is =%s" % (ytested[i]))
```

```
Using TensorFlow backend.
```

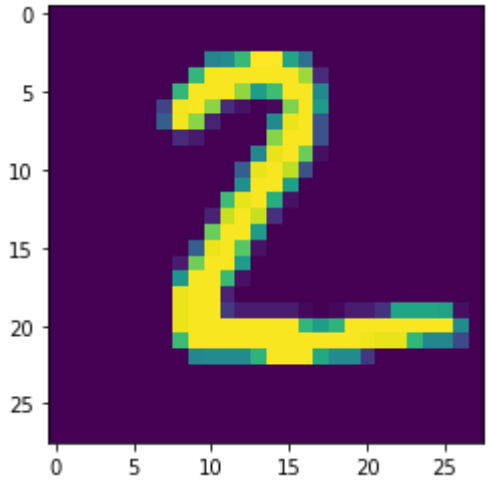
The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:79

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:57: UserWarning: The `nb
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

Epoch 1/10
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

60000/60000 [==============================] - 28s 468us/step - loss: 0.2044 - acc: 0
Epoch 2/10
60000/60000 [==============================] - 21s 348us/step - loss: 0.0837 - acc: 0
Epoch 3/10
60000/60000 [==============================] - 21s 345us/step - loss: 0.0651 - acc: 0
Epoch 4/10
60000/60000 [==============================] - 21s 348us/step - loss: 0.0550 - acc: 0
Epoch 5/10
60000/60000 [==============================] - 21s 346us/step - loss: 0.0472 - acc: 0
Epoch 6/10
60000/60000 [==============================] - 21s 350us/step - loss: 0.0420 - acc: 0
Epoch 7/10
60000/60000 [==============================] - 21s 352us/step - loss: 0.0379 - acc: 0
Epoch 8/10
60000/60000 [==============================] - 21s 350us/step - loss: 0.0336 - acc: 0
Epoch 9/10
```

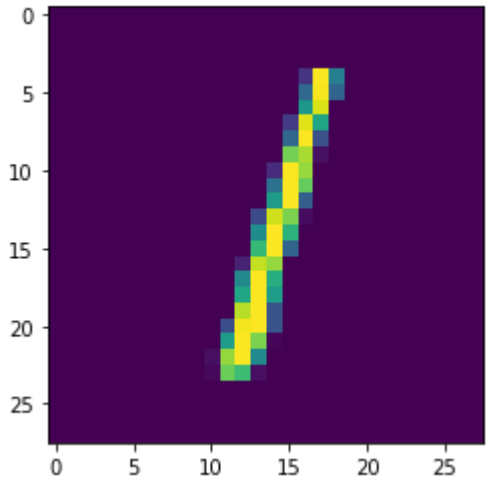
```
60000/60000 [==============================] - 21s 350us/step - loss: 0.0316 - acc: 0
Epoch 10/10
60000/60000 [==============================] - 21s 349us/step - loss: 0.0306 - acc: 0
10000/10000 [==============================] - 1s 115us/step
Testing accuracy - >  99.14
```



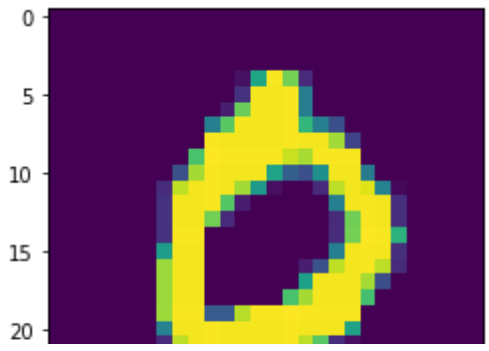
```
The Predicted Testing image is =7
```

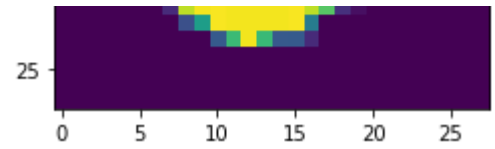


```
The Predicted Testing image is =2
```



```
The Predicted Testing image is =1
```






```
The Predicted Testing image is =0
```

```
3+4
```

```
score
```

```
score[1] * 100
```

```
# 3. Import libraries and modules
import numpy as np
np.random.seed(123)  # for reproducibility

from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.utils import np_utils
from keras.datasets import mnist

# 4. Load pre-shuffled MNIST data into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# 5. Preprocess input data
img_rows = X_train.shape[1]
img_cols = X_train.shape[2]

X_train = X_train.reshape(X_train.shape[0], img_cols, img_rows, 1)
X_test = X_test.reshape(X_test.shape[0], img_cols, img_rows, 1)
#X_train = X_train.reshape(X_train.shape[0], 1, 28, 28)
#X_test = X_test.reshape(X_test.shape[0], 1, 28, 28)
#X_train = X_train.astype('float32')
#X_test = X_test.astype('float32')
X_train = X_train / 255
X_test = X_test /255

# 6. Preprocess class labels
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)

# 7. Define model architecture
model = Sequential()

model.add(Convolution2D(32, 3, 3, activation='relu', input_shape=(1,28,28)))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
```

```
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

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

# 9. Fit model on training data
model.fit(X_train, Y_train,
          batch_size=32, nb_epoch=10, verbose=1)

# 10. Evaluate model on test data
score = model.evaluate(X_test, Y_test, verbose=0)
```

```
import matplotlib.pyplot as plt
#plot the first image in the dataset
plt.imshow(X_train[0])
#check image shape
X_train[0].shape

#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
```

```
from keras.utils import to_categorical
#one-hot encode target column
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]

#X_train = X_train.reshape(60000,28,28,1)
#X_test = X_test.reshape(10000,28,28,1)
#X_train = X_train.reshape((-1, 1, 100, 1))
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten
#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))

#compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

#train the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3)

#predict first 4 images in the test set
model.predict(X_test[:4])

#actual results for first 4 images in test set
y_test[:4]
```