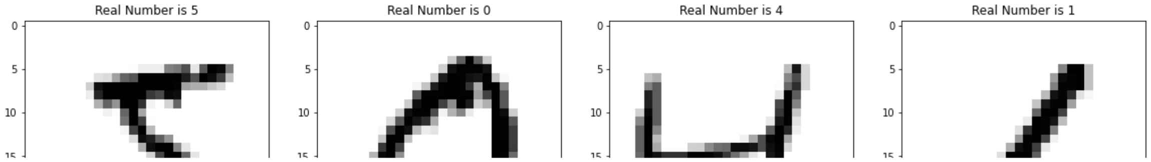
In [10]:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import keras
from keras.models import Sequential
from keras.layers import Conv2D, Lambda, MaxPooling2D
from keras.layers import Dense, Dropout, Flatten

from keras.layers import  BatchNormalization
from keras.callbacks import *

from matplotlib import pyplot as plt
from keras.utils.np_utils import to_categorical


####import data#######
# Load training and eval data
((train_data, train_labels),
 (eval_data, eval_labels)) = tf.keras.datasets.mnist.load_data()
train_labels = train_labels.astype(np.int32)
eval_labels = eval_labels.astype(np.int32)

############show the images with real number#########3
fig, axis = plt.subplots(1, 4, figsize=(20, 10))
for i, ax in enumerate(axis.flat):
    ax.imshow(train_data[i], cmap='binary')
    digit = train_labels[i]
    ax.set(title = f"Real Number is {digit}");

############## reshape train and validation data to prepare for learning model
train_data=train_data.reshape(-1,28,28,1)
eval_data=eval_data.reshape(-1,28,28,1)

a = np.array(train_labels)
train_labels = tf.keras.utils.to_categorical(a-1, num_classes = 10)

b=np.array(eval_labels)
eval_labels = tf.keras.utils.to_categorical(b-1,num_classes=10)


#################### creation of a deep learning model based on the keras frame

model=Sequential()

model.add(Conv2D(filters=64, kernel_size = (3,3), activation="relu", input_sha
model.add(Conv2D(filters=64, kernel_size = (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=128, kernel_size = (3,3), activation="relu"))
model.add(Conv2D(filters=128, kernel_size = (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=256, kernel_size = (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
#Dense layer that flatten the output of the CNN
model.add(Flatten())

#Dense Layer
model.add(Dense(512,activation="relu"))

model.add(Dense(10,activation="softmax"))

#Add loss function, metrics, optimizer
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["acc

#Adding callbacks
mc=ModelCheckpoint('best_model_handwritten.h5', monitor='val_acc', mode='max',
#Print summary of model
model.summary()

##############Training the model with training data####################
print('Starting....')
history=model.fit(train_data, train_labels, batch_size=128, validation_split=0
```

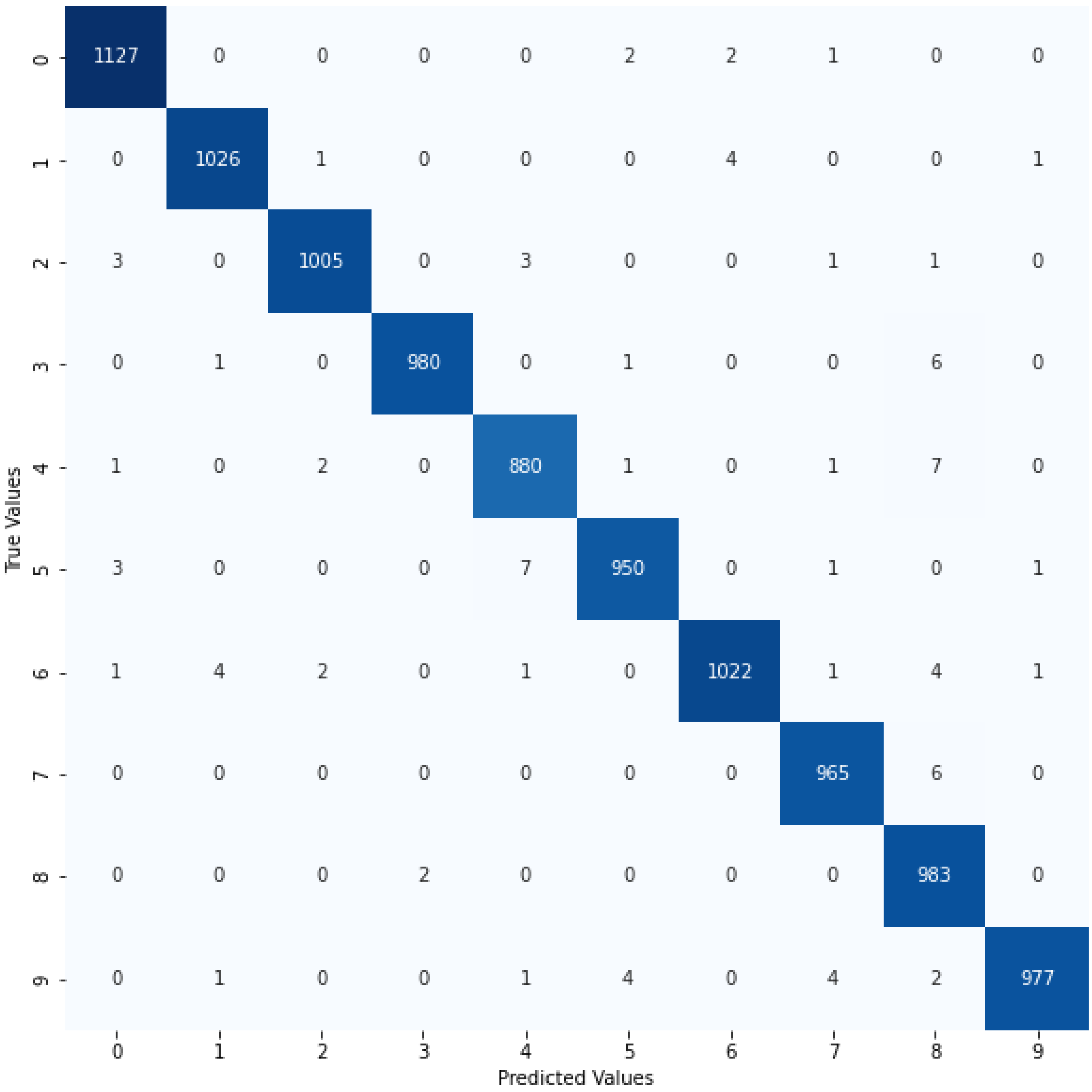
```
e, skipping.
375/375 [==============================] - 67s 180ms/step - loss: 0.0101 -
accuracy: 0.9965 - val_loss: 0.0516 - val_accuracy: 0.9882
Epoch 9/10
375/375 [==============================] - ETA: 0s - loss: 0.0128 - accurac
y: 0.9963WARNING:tensorflow:Can save best model only with val_acc availabl
e, skipping.
375/375 [==============================] - 67s 179ms/step - loss: 0.0128 -
accuracy: 0.9963 - val_loss: 0.0417 - val_accuracy: 0.9908
Epoch 10/10
375/375 [==============================] - ETA: 0s - loss: 0.0097 - accurac
y: 0.9969WARNING:tensorflow:Can save best model only with val_acc availabl
e, skipping.
375/375 [==============================] - 67s 179ms/step - loss: 0.0097 -
accuracy: 0.9969 - val_loss: 0.0500 - val_accuracy: 0.9908
```



In [11]:

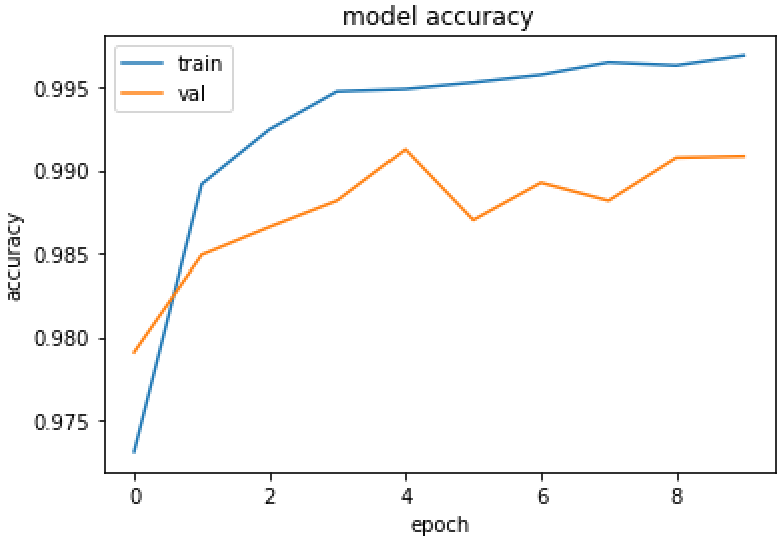
```
########################## Evaluation testing data ########################
score=model.evaluate(eval_data,eval_labels,verbose=0)
print('loss: ', score[0])
print('accuracy: ', score[1])

fig = plt.figure(figsize=(10, 10)) # Set Figure

y_pred = model.predict(eval_data) # Predict class probabilities

Y_pred = np.argmax(y_pred, 1) # Decode Predicted labels
Y_test = np.argmax(eval_labels, 1) # Decode labels frome one hot foemat

mat = confusion_matrix(Y_test, Y_pred) # Confusion matrix

########################## Plot confusion matrix ############################
sns.heatmap(mat.T, square=True, annot=True, cbar=False, cmap=plt.cm.Blues, fmt
plt.xlabel('Predicted Values')
plt.ylabel('True Values');
plt.show();
```

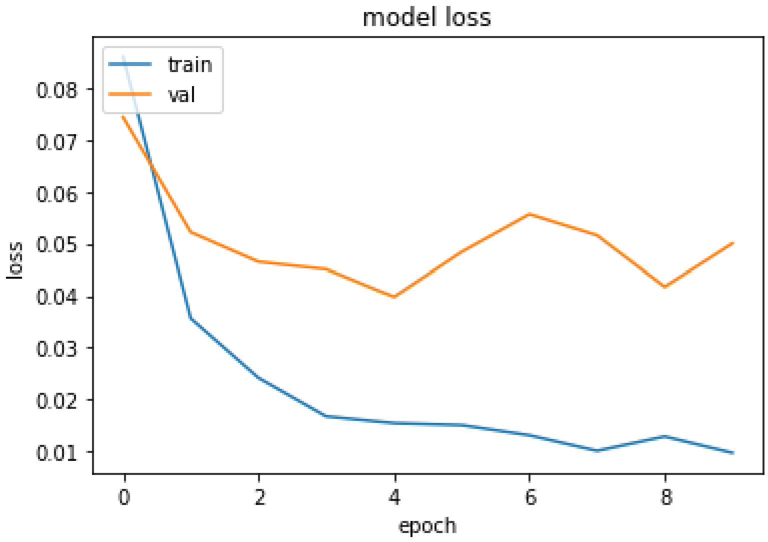
```
loss:  0.03694010153412819
accuracy:  0.9915000200271606
```



	0	1	2	3	4	5	6	7	8	9
0	1127	0	0	0	0	2	2	1	0	0
1	0	1026	1	0	0	0	4	0	0	1
2	3	0	1005	0	3	0	0	1	1	0
3	0	1	0	980	0	1	0	0	6	0
4	1	0	2	0	880	1	0	1	7	0
5	3	0	0	0	7	950	0	1	0	1
6	1	4	2	0	1	0	1022	1	4	1
7	0	0	0	0	0	0	0	965	6	0
8	0	0	0	2	0	0	0	0	983	0
9	0	1	0	0	1	4	0	4	2	977

```
In [12]: ########################### Plot result ###############################
         plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(['train', 'val'], loc='upper left')
         plt.show()
         #Loss result
         plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title('model loss')
         plt.ylabel('loss')
         plt.xlabel('epoch')
         plt.legend(['train', 'val'], loc='upper left')
         plt.show()
```





```
In [ ]:
```