Capstone Project

April 17, 2021

1 Capstone Project

1.1 Image classifier for the SVHN dataset

1.1.1 Instructions

In this notebook, you will create a neural network that classifies real-world images digits. You will use concepts from throughout this course in building, training, testing, validating and saving your Tensorflow classifier model.

This project is peer-assessed. Within this notebook you will find instructions in each section for how to complete the project. Pay close attention to the instructions as the peer review will be carried out according to a grading rubric that checks key parts of the project instructions. Feel free to add extra cells into the notebook as required.

1.1.2 How to submit

When you have completed the Capstone project notebook, you will submit a pdf of the notebook for peer review. First ensure that the notebook has been fully executed from beginning to end, and all of the cell outputs are visible. This is important, as the grading rubric depends on the reviewer being able to view the outputs of your notebook. Save the notebook as a pdf (File -> Download as -> PDF via LaTeX). You should then submit this pdf for review.

1.1.3 Let's get started!

We'll start by running some imports, and loading the dataset. For this project you are free to make further imports throughout the notebook as you wish.

```
In [42]: import tensorflow as tf
         from scipy.io import loadmat
         import numpy as np
         import matplotlib.pylab as plt

         #%matpotlib inline
```

For the capstone project, you will use the SVHN dataset. This is an image dataset of over 600,000 digit images in all, and is a harder dataset than MNIST as the numbers appear in the context of natural scene images. SVHN is obtained from house numbers in Google Street View images.

- Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng. "Reading Digits in Natural Images with Unsupervised Feature Learning". NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011.

Your goal is to develop an end-to-end workflow for building, training, validating, evaluating and saving a neural network that classifies a real-world image into one of ten classes.

```
In [43]: # Run this cell to load the dataset

         train = loadmat('data/train_32x32.mat')
         test = loadmat('data/test_32x32.mat')
```

Both `train` and `test` are dictionaries with keys `X` and `y` for the input images and labels respectively.

1.2 1. Inspect and preprocess the dataset

- Extract the training and testing images and labels separately from the train and test dictionaries loaded for you.
- Select a random sample of images and corresponding labels from the dataset (at least 10), and display them in a figure.
- Convert the training and test images to grayscale by taking the average across all colour channels for each pixel. *Hint: retain the channel dimension, which will now have size 1.*
- Select a random sample of the grayscale images and corresponding labels from the dataset (at least 10), and display them in a figure.

```
In [44]: x_train, y_train, x_test, y_test = train['X'], train['y'], test['X'], test['y']
```

```
In [45]: x_train = x_train / 255.0 # scale images
         x_test = x_test / 255.0
```

```
In [46]: x_train = np.transpose(x_train, (3, 0, 1, 2)) # transpose
         x_test = np.transpose(x_test, (3, 0, 1, 2))

In [47]: y_train = np.ravel(y_train) % 10 # label values should be in [0,10)
         y_test = np.ravel(y_test) % 10

In [124]: def plot_images(x_train, y_train, indices, nchannel=3):
              plt.figure(figsize=(20,20))
              for i in range(n):
                  plt.subplot(10, 10, i+1)
                  index = indices[i]
                  if nchannel == 3:
                      plt.imshow(x_train[index, ...])
                  else:
                      plt.imshow(x_train[index, :, :, 0], cmap='gray')
                  plt.title(f'label: {y_train[index]}', size=15)
                  plt.axis('off')
              plt.show()

          n = 100
          indices = np.random.choice(len(y_train), n)
          plot_images(x_train, y_train, indices)
```


```
In [48]: x_train = np.mean(x_train, axis=-1, keepdims=True)
         x_test = np.mean(x_test, axis=-1, keepdims=True)

In [126]: plot_images(x_train, y_train, indices, nchannel=1)
```

1.3 2. MLP neural network classifier

- Build an MLP classifier model using the Sequential API. Your model should use only Flatten and Dense layers, with the final layer having a 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different MLP architectures. *Hint: to achieve a reasonable accuracy you won't need to use more than 4 or 5 layers.*
- Print out the model summary (using the summary() method)
- Compile and train the model (we recommend a maximum of 30 epochs), making use of both training and validation sets during the training run.
- Your model should track at least one appropriate metric, and use at least two callbacks during training, one of which should be a ModelCheckpoint callback.
- As a guide, you should aim to achieve a final categorical cross entropy training loss of less than 1.0 (the validation loss might be higher).

- Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.
- Compute and display the loss and accuracy of the trained model on the test set.

```
In [60]: from tensorflow.keras.models import Sequential
         from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, BatchNormali

         def get_model_MLP(input_shape):
             return Sequential([
                 Flatten(input_shape=input_shape, name='flatten'),
                 Dense(1024, kernel_initializer='he_uniform', bias_initializer="ones", activati
                 Dense(512, activation="relu", name='layer2'),
                 Dense(10, activation='softmax', name='final')
             ],
             name='sequential')

In [61]: model = get_model_MLP(x_train.shape[1:])

In [62]: model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
layer1 (Dense)               (None, 1024)              1049600
_________________________________________________________________
layer2 (Dense)               (None, 512)               524800
_________________________________________________________________
final (Dense)                (None, 10)                5130
=================================================================
Total params: 1,579,530
Trainable params: 1,579,530
Non-trainable params: 0
_________________________________________________________________


In [52]: def compile_model(model):
             model.compile(optimizer='adam',
                           loss='sparse_categorical_crossentropy',
                           metrics=['accuracy'])

         compile_model(model)

In [53]: from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

         def get_checkpoint_every_epoch(prefix='MLP_'):
             return ModelCheckpoint(prefix + 'checkpoints_every_epoch/checkpoint_{epoch:02d}',
```

```
        def get_checkpoint_best_only(prefix='MLP_'):
            return ModelCheckpoint(prefix + 'checkpoints_best_only/checkpoint', save_weights_
                                   monitor='val_accuracy', mode='max')

        def get_early_stopping():
            return EarlyStopping(monitor='val_accuracy', patience=3)

        checkpoint_every_epoch = get_checkpoint_every_epoch()
        checkpoint_best_only = get_checkpoint_best_only()
        early_stopping = get_early_stopping()
```

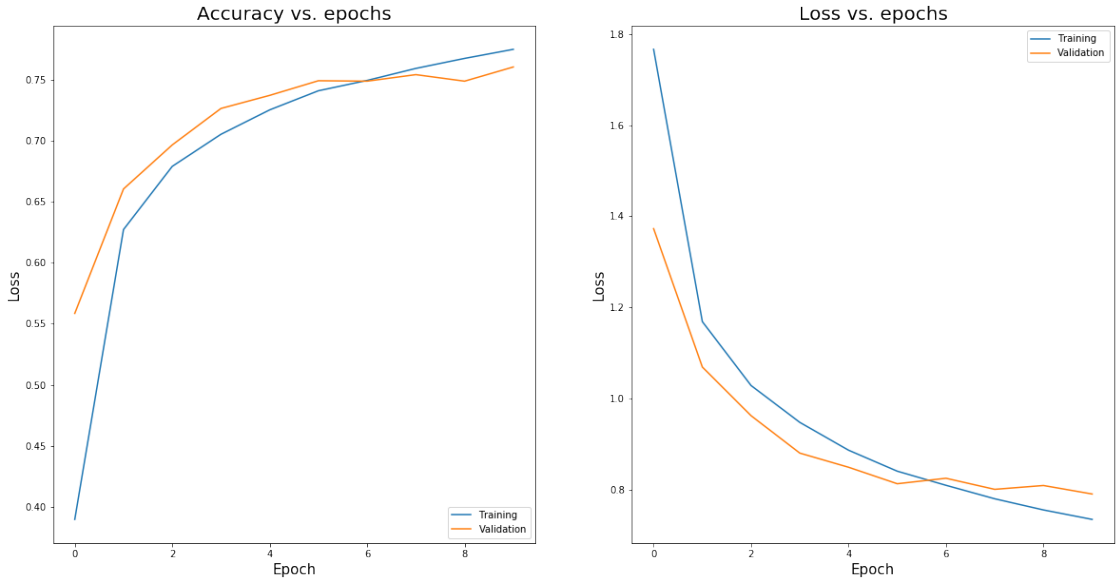
```
In [54]: def train_model(model, x_train, y_train, epochs, callbacks):
             return model.fit(x_train, y_train, epochs=epochs, validation_split=0.15, callbacks

         callbacks = [checkpoint_every_epoch, checkpoint_best_only, early_stopping]
         history = train_model(model, x_train, y_train, epochs=10, callbacks=callbacks)
```

```
Train on 62268 samples, validate on 10989 samples
Epoch 1/10
62268/62268 [==============================] - 127s 2ms/sample - loss: 1.7663 - accuracy: 0.389
Epoch 2/10
62268/62268 [==============================] - 119s 2ms/sample - loss: 1.1687 - accuracy: 0.627
Epoch 3/10
62268/62268 [==============================] - 119s 2ms/sample - loss: 1.0284 - accuracy: 0.679
Epoch 4/10
62268/62268 [==============================] - 118s 2ms/sample - loss: 0.9476 - accuracy: 0.705
Epoch 5/10
62268/62268 [==============================] - 118s 2ms/sample - loss: 0.8866 - accuracy: 0.725
Epoch 6/10
62268/62268 [==============================] - 118s 2ms/sample - loss: 0.8405 - accuracy: 0.741
Epoch 7/10
62268/62268 [==============================] - 118s 2ms/sample - loss: 0.8095 - accuracy: 0.749
Epoch 8/10
62268/62268 [==============================] - 117s 2ms/sample - loss: 0.7799 - accuracy: 0.759
Epoch 9/10
62268/62268 [==============================] - 118s 2ms/sample - loss: 0.7554 - accuracy: 0.767
Epoch 10/10
62268/62268 [==============================] - 117s 2ms/sample - loss: 0.7347 - accuracy: 0.775
```

```
In [55]: def plot_accuracy_loss(history):
             plt.figure(figsize=(20,10))
             plt.subplot(1,2,1)
             plt.plot(history.history['accuracy'])
             plt.plot(history.history['val_accuracy'])
             plt.title('Accuracy vs. epochs', size=20)
             plt.ylabel('Loss', size=15)
```

```
        plt.xlabel('Epoch', size=15)
        plt.legend(['Training', 'Validation'], loc='lower right')
        plt.subplot(1,2,2)
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title('Loss vs. epochs', size=20)
        plt.ylabel('Loss', size=15)
        plt.xlabel('Epoch', size=15)
        plt.legend(['Training', 'Validation'], loc='upper right')
        plt.show()
```

In [56]:
```
plot_accuracy_loss(history)
```



In [57]:
```
def get_test_accuracy(model, x_test, y_test):
        test_loss, test_acc = model.evaluate(x=x_test, y=y_test, verbose=0)
        print('test accuracy: {acc:0.3f}'.format(acc=test_acc))
```

In [58]:
```
get_test_accuracy(model, x_test, y_test)
```

```
test accuracy: 0.740
```

In [59]:
```
!ls
```

```
'Capstone Project.ipynb'   MLP_checkpoints_best_only
 data                      MLP_checkpoints_every_epoch
```

1.4 3. CNN neural network classifier

- Build a CNN classifier model using the Sequential API. Your model should use the Conv2D, MaxPool2D, BatchNormalization, Flatten, Dense and Dropout layers. The final layer should again have a 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different CNN architectures. *Hint: to achieve a reasonable accuracy you won't need to use more than 2 or 3 convolutional layers and 2 fully connected layers.)*
- The CNN model should use fewer trainable parameters than your MLP model.
- Compile and train the model (we recommend a maximum of 30 epochs), making use of both training and validation sets during the training run.
- Your model should track at least one appropriate metric, and use at least two callbacks during training, one of which should be a ModelCheckpoint callback.
- You should aim to beat the MLP model performance with fewer parameters!
- Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.
- Compute and display the loss and accuracy of the trained model on the test set.

```
In [67]: def get_model_CNN(input_shape, dropout_rate=0.3):
             return Sequential([
                 Conv2D(32, (3,3), kernel_initializer='he_uniform', bias_initializer="ones", a
                 BatchNormalization(),
                 MaxPooling2D((2,2), name='pool1'),
                 Dropout(0.2, name='drop1'),
                 Conv2D(64, (3,3), activation='relu', input_shape=input_shape, name='conv2'),
                 BatchNormalization(),
                 MaxPooling2D((2,2), name='pool2'),
                 Dropout(0.2, name='drop2'),
                 Flatten(name='flatten'),
                 Dense(64, activation='relu', name='dense2'),
                 Dropout(0.25, name='drop3'),
                 Dense(10, activation='softmax', name='final')
             ], name='conv')

In [68]: model = get_model_CNN(x_train.shape[1:])

In [69]: model.summary()

Model: "conv"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               (None, 30, 30, 32)        320
_________________________________________________________________
batch_normalization_11 (Batc (None, 30, 30, 32)        128
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 15, 15, 32)        0
_________________________________________________________________
drop1 (Dropout)              (None, 15, 15, 32)        0
```

```
_________________________________________________________________
conv2 (Conv2D)               (None, 13, 13, 64)        18496
_________________________________________________________________
batch_normalization_12 (Batc (None, 13, 13, 64)        256
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 6, 6, 64)          0
_________________________________________________________________
drop2 (Dropout)              (None, 6, 6, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
dense2 (Dense)               (None, 64)                147520
_________________________________________________________________
drop3 (Dropout)              (None, 64)                0
_________________________________________________________________
final (Dense)                (None, 10)                650
=================================================================
Total params: 167,370
Trainable params: 167,178
Non-trainable params: 192
_________________________________________________________________
```

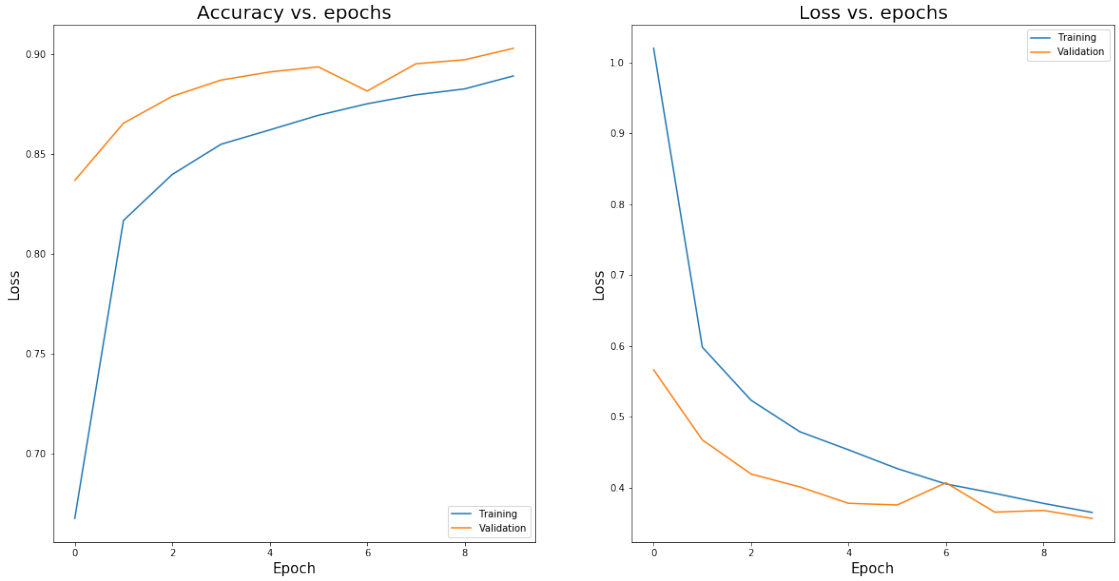
```
In [26]: compile_model(model)
```

```
In [27]: checkpoint_every_epoch = get_checkpoint_every_epoch(prefix='CONV_')
         checkpoint_best_only = get_checkpoint_best_only(prefix='CONV_')
         early_stopping = get_early_stopping()
         callbacks = [checkpoint_every_epoch, checkpoint_best_only, early_stopping]
         history = train_model(model, x_train, y_train, epochs=10, callbacks=callbacks)
```

```
Train on 62268 samples, validate on 10989 samples
Epoch 1/10
62268/62268 [==============================] - 541s 9ms/sample - loss: 1.0197 - accuracy: 0.667
Epoch 2/10
62268/62268 [==============================] - 534s 9ms/sample - loss: 0.5981 - accuracy: 0.816
Epoch 3/10
62268/62268 [==============================] - 533s 9ms/sample - loss: 0.5233 - accuracy: 0.839
Epoch 4/10
62268/62268 [==============================] - 537s 9ms/sample - loss: 0.4789 - accuracy: 0.855
Epoch 5/10
62268/62268 [==============================] - 538s 9ms/sample - loss: 0.4534 - accuracy: 0.862
Epoch 6/10
62268/62268 [==============================] - 539s 9ms/sample - loss: 0.4269 - accuracy: 0.869
Epoch 7/10
62268/62268 [==============================] - 538s 9ms/sample - loss: 0.4052 - accuracy: 0.875
Epoch 8/10
62268/62268 [==============================] - 546s 9ms/sample - loss: 0.3920 - accuracy: 0.879
```

```
Epoch 9/10
62268/62268 [==============================] - 545s 9ms/sample - loss: 0.3779 - accuracy: 0.882
Epoch 10/10
62268/62268 [==============================] - 544s 9ms/sample - loss: 0.3651 - accuracy: 0.889
```

In [28]:
```
plot_accuracy_loss(history)
```



In [199]:
```
#def get_model_last_epoch(model):
#    model.load_weights(tf.train.latest_checkpoint('CONV_checkpoints_every_epoch'))
#    return model

#model = get_model_CNN(x_train.shape[1:])
#compile_model(model)
#model = get_model_last_epoch(model)
```

In [29]:
```
get_test_accuracy(model, x_test, y_test)
```

```
test accuracy: 0.888
```

In [30]:
```
!ls -ltr
```

```
total 952
drwxrwxrwx 2 nobody nogroup   6144 Apr 15 09:10  data
drwxr-xr-x 2 jovyan users     6144 Apr 17 19:27  MLP_checkpoints_every_epoch
drwxr-xr-x 2 jovyan users     6144 Apr 17 19:27  MLP_checkpoints_best_only
drwxr-xr-x 2 jovyan users     6144 Apr 17 21:03  CONV_checkpoints_every_epoch
drwxr-xr-x 2 jovyan users     6144 Apr 17 21:03  CONV_checkpoints_best_only
-rwxrwxrwx 1 nobody nogroup 952482 Apr 17 21:05 'Capstone Project.ipynb'
```

1.5 4. Get model predictions

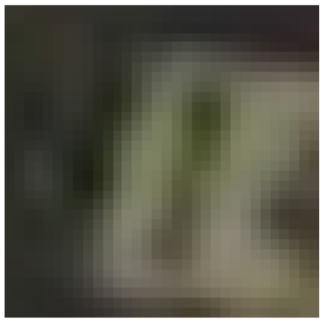
- Load the best weights for the MLP and CNN models that you saved during the training run.
- Randomly select 5 images and corresponding labels from the test set and display the images with their labels.
- Alongside the image and label, show each model's predictive distribution as a bar chart, and the final model prediction given by the label with maximum probability.

```
In [31]: x_test, y_test = test['X'], test['y']
         x_test = x_test / 255.0
         x_test = np.transpose(x_test, (3, 0, 1, 2))
         y_test = np.ravel(y_test) % 10

In [32]: np.random.seed(1)
         n = 5 # choose 5 samples from test images
         indices = np.random.choice(len(y_test), n)

In [33]: def get_model_best_epoch_model(model, prefix='MLP_'):
             model.load_weights(prefix + 'checkpoints_best_only/checkpoint')
             return model

In [34]: def plot_images_bars(model, x_test, y_test, indices):
             n = len(indices)
             plt.figure(figsize=(10,20))
             for i in range(n):
                 plt.subplot(n, 2, 2*i+1)
                 index = indices[i]
                 plt.imshow(x_test[index, ...])
                 plt.title(f'label: {y_test[index]}', size=15)
                 plt.axis('off')
                 plt.subplot(n, 2, 2*i+2)
                 x_test_gray = np.mean(x_test[index, ...], axis=-1, keepdims=True)
                 p = model.predict(x_test_gray[np.newaxis, ...]).flatten()
                 plt.bar(range(len(p)), p)
                 plt.xticks(range(10))
                 plt.title(f'predicted: {np.argmax(p)}')
             plt.show()
```

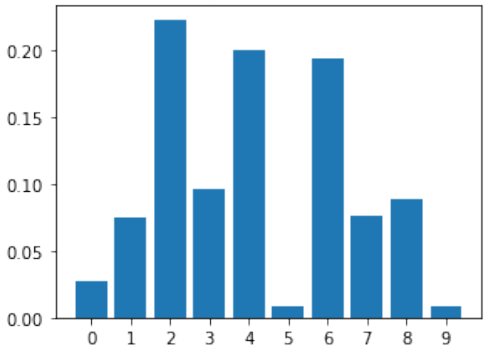
1.6 Prediction with MLP

```
In [35]: model = get_model_MLP(x_train.shape[1:])
         model = get_model_best_epoch_model(model)

In [36]: plot_images_bars(model, x_test, y_test, indices)
```

label: 1

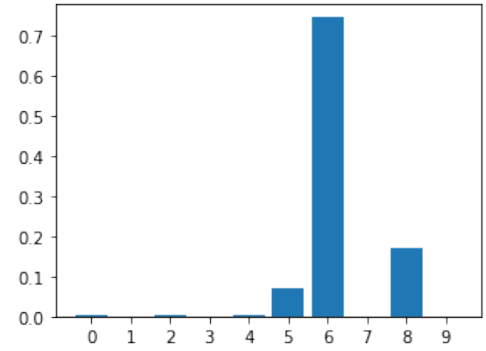


predicted: 2



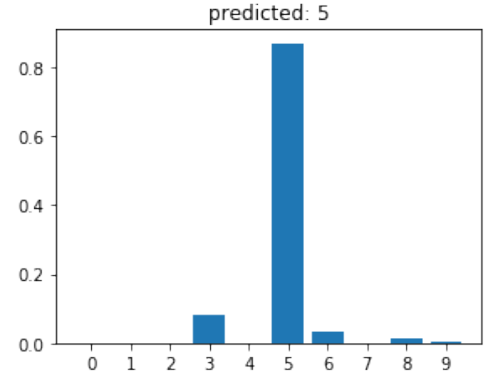
label: 6

predicted: 6



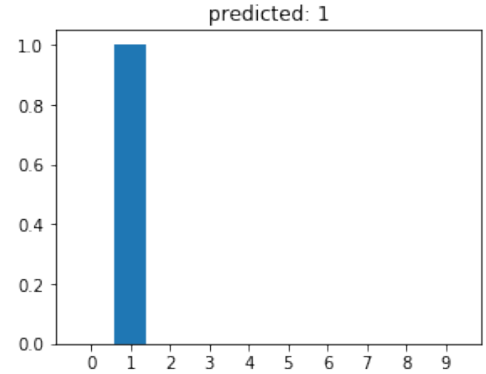
label: 5

predicted: 5

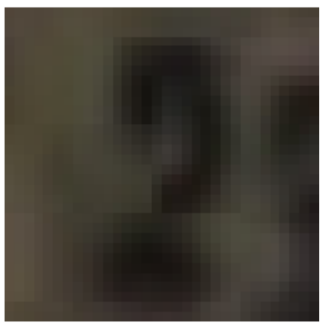


label: 1

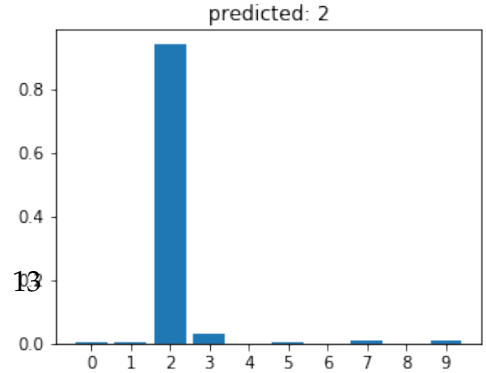
predicted: 1



label: 2



predicted: 2

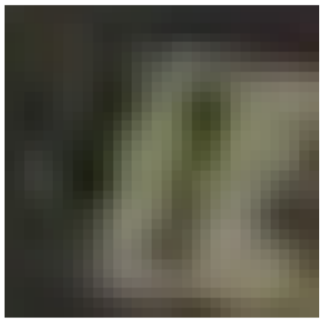


1.7 Prediction with CNN

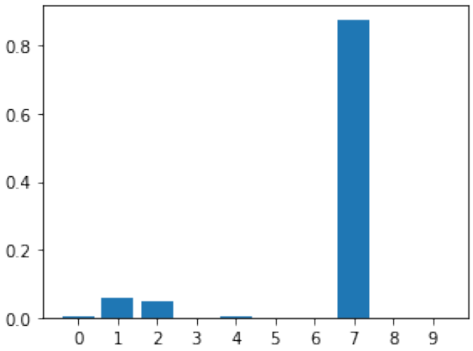
```
In [37]: model = get_model_CNN(x_train.shape[1:])
         model = get_model_best_epoch_model(model, prefix='CONV_')
```

```
In [39]: plot_images_bars(model, x_test, y_test, indices)
```

label: 1

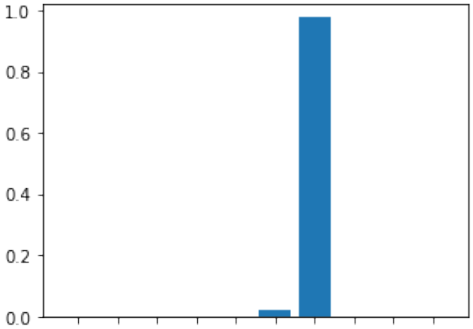


predicted: 7



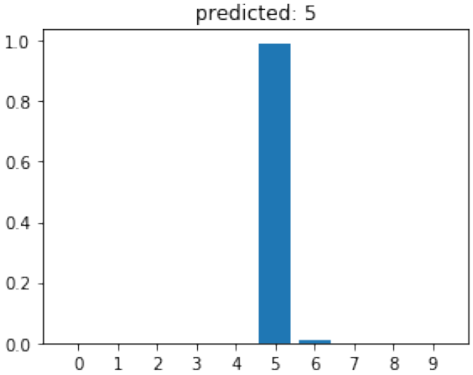
label: 6

predicted: 6



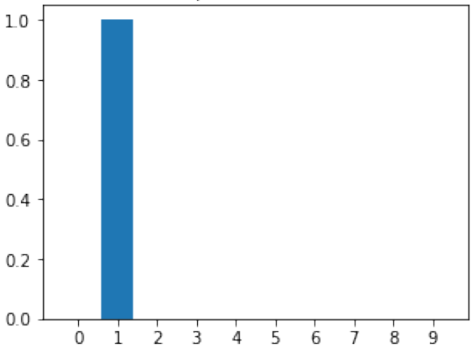
label: 5

predicted: 5

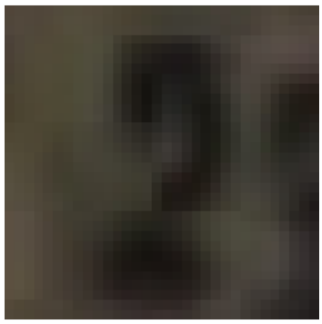


label: 1

predicted: 1



label: 2



predicted: 2