SUMMER INTERNSHIP REPORT

Area of Online Internship	Machine/Deep Learning
Intern Name	Aditi Ashish Gawande
Name Of Institution	INDIAN INSTITUTE OF TECHNOLOGY, INDORE
Faculty Mentor Name	Dr. Vimal Bhatia
Duration	2 MONTHS (5/11/2021 TO 29/12/2021)
Date Of Submission	29/12/2021

TABLE OF CONTENTS

CONVOLUTIONAL NEURAL NETWORK(CNN)

A convolutional neural network, or CNN, is a deep learning neural network designed for processing structured arrays of data such as images. Convolutional neural networks are widely used in computer vision and have become the state of the art for many visual applications such as image classification, and have also found success in natural language processing for text classification.
Convolutional neural networks are very good at picking up on patterns in the input image, such as lines, gradients, circles, or even eyes and faces. It is this property that makes convolutional neural networks so powerful for computer vision. Unlike earlier computer vision algorithms, convolutional neural networks can operate directly on a raw image and do not need any preprocessing.
A convolutional neural network is a feed-forward neural network, often with up to 20 or 30 layers. The power of a convolutional neural network comes from a special kind of layer called the convolutional layer.
Convolutional neural networks contain many convolutional layers stacked on top of each other, each one capable of recognizing more sophisticated shapes. With three or four convolutional layers it is possible to recognize handwritten digits and with 25 layers it is possible to distinguish human faces.
The usage of convolutional layers in a convolutional neural network mirrors the structure of the human visual cortex, where a series of layers process an incoming image and identify progressively more complex features.

Convolutional Neural Network Design

The architecture of a convolutional neural network is a multi-layered feed-forward neural network, made by stacking many hidden layers on top of each other in sequence. It is this sequential design that allows convolutional neural networks to learn hierarchical features.
The hidden layers are typically convolutional layers followed by activation layers, some of them followed by pooling layers.
A simple convolutional neural network that aids understanding of the core design principles is the early convolutional neural network LeNet-5, published by Yann LeCun in 1998. LeNet is capable of recognizing handwritten characters.

Example Convolutional Neural Network Layers Explained

LeNet takes an input image of a handwritten digit of size 32x32 pixels and passes it through a stack of the following layers. Each layer except the last is followed by a tanh activation function:

C1	

	The first convolutional layer. This consists of six convolutional kernels of size 5x5, which ‘walk over’ the input image. C1 outputs six images of size 28x28. The first layer of a convolutional neural network normally identifies basic features such as straight edges and corners.
S2	A subsampling layer, also known as an average pooling layer. Each square of four pixels in the C1 output is averaged to a single pixel. S2 scales down the six 28x28 images by a factor of 2, producing six output images of size 14x14.
C3	The second convolutional layer. This consists of 16 convolutional kernels, each of size 5x5, which take the six 14x14 images and walk over them again, producing 16 images of size 10x10.
S4	The second average pooling layer. S4 scales down the sixteen 10x10 images to sixteen 5x5 images.
C5	A fully connected convolutional layer with 120 outputs. Each of the 120 output nodes is connected to all of the 400 nodes (5x5x16) that came from S4. At this point the output is no longer an image, but a 1D array of length 120.
F6	A fully connected layer mapping the 120-array to a new array of length 10. Each element of the array now corresponds to a handwritten digit 0-9.
Output Layer	A softmax function which transforms the output of F6 into a probability distribution of 10 values which sum to 1.

LeNet-5 is one of the simplest convolutional neural networks, with six layers. This gives it enough power to distinguish small handwritten digits but not, for example, the 26 letters of the alphabet, and especially not faces or objects. Today the most sophisticated networks may have more than 30 layers and millions of parameters, and also involve branching, however the basic building blocks of convolutional kernels remain the same.

Convolutional Layer

The key building block in a convolutional neural network is the convolutional layer. We can visualize a convolutional layer as many small square templates, called convolutional kernels, which slide over the image and look for patterns. Where that part of the image matches the kernel's pattern, the kernel returns a large positive value, and when there is no match, the kernel returns zero or a smaller value.

Example Calculation of a Convolution on a Matrix

Mathematically, the kernel is a matrix of weights. For example, the following 3x3 kernel detects vertical lines.

Let us imagine an 9x9 input image of a plus sign. This has two kinds of lines, horizontal and vertical, and a crossover. In matrix format the image would look as follows:

Imagine we want to test the vertical line detector kernel on the plus sign image. To perform the convolution, we slide the convolution kernel over the image. At each position, we multiply each element of the convolution kernel by the element of the image that it covers, and sum the results.

Since the kernel has width 3, it can only be positioned at 7 different positions horizontally in an image of width 9. So the end result of the convolution operation on an image of size 9x9 with a 3x3 convolution kernel is a new image of size 7x7.

So in the above example, first the kernel is placed in the top left corner and each element of the kernel is multiplied by each element in the red box in the top left of the original image. Since these values are all 0, the result for that cell is 0 in the top left of the output matrix.

Now let us consider the position of the blue box in the above example. It contains part of a vertical line. When the kernel is placed over this vertical line, it matches and returns 3.

Recall that this convolution kernel is a vertical line detector. For the parts of the original image which contained a vertical line, the kernel has returned a value 3,

whereas it has returned a value of 1 for the horizontal line, and 0 for the empty areas of the image.
In practice, a convolution kernel contains both weights and biases, similar to the formula for linear regression. So an input pixel is multiplied by the weight and then the bias is added.

Example of Convolution on a Image

Let us consider the following 9x9 convolution kernel, which is a slightly more sophisticated vertical line detector than the kernel used in the last example:

And we can take the following image of a tabby cat with dimensions 204x175, which we can represent as a matrix with values in the range from 0 to 1, where 1 is white and 0 is black.

Applying the convolution, we find that the filter has performed a kind of vertical line detection. The vertical stripes on the tabby cat's head are highlighted in the output. The output image is 8 pixels smaller in both dimensions due to the size of the kernel (9x9).

Despite its simplicity, the ability to detect vertical or horizontal lines, corners, curves, and other simple features, is an extremely powerful property of the convolution kernel. We recall that a convolutional layer is made up of a series of convolution kernels. Typically, the first layer of a convolutional neural network contains a vertical line detector, a horizontal line detector, and various diagonal, curve and corner detectors. These feature detector kernels are not programmed by a human but in fact are learned by the neural network during training, and serve as the first stage of the image recognition process.
Later layers in the neural network are able to build on the features detected by earlier layers and identify ever more complex shapes.

Applications of Convolutional Neural Networks

Convolutional neural networks are most widely known for image analysis but they have also been adapted for several applications in other areas of machine learning, such as natural language processing.

Convolutional Neural Networks for Self-Driving Cars

Several companies, such as Tesla and Uber, are using convolutional neural networks as the computer vision component of a self-driving car.

A self-driving car's computer vision system must be capable of localization, obstacle avoidance, and path planning.
Let us consider the case of pedestrian detection. A pedestrian is a kind of obstacle which moves. A convolutional neural network must be able to identify the location of the pedestrian and extrapolate their current motion in order to calculate if a collision is imminent.
A convolutional neural network for object detection is slightly more complex than a classification model, in that it must not only classify an object, but also return the four coordinates of its bounding box.
Furthermore, the convolutional neural network designer must avoid unnecessary false alarms for irrelevant objects, such as litter, but also take into account the high cost of miscategorizing a true pedestrian and causing a fatal accident.
A major challenge for this kind of use is collecting labeled training data. Google's Captcha system is used for authenticating on websites, where a user is asked to categorize images as fire hydrants, traffic lights, cars, etc. This is actually a useful way to collect labeled training images for purposes such as self-driving cars and Google StreetView.

Convolutional Neural Networks for Text Classification

Although convolutional neural networks were initially conceived as a computer vision tool, they have been adapted for the field of natural language processing with great success.
The principle behind their use on text is very similar to the process for images, with the exception of a preprocessing stage. To use a convolutional neural network for text classification, the input sentence is tokenized and then converted into an array of word vector embeddings using a lookup such as word2vec. It is then passed through a convolutional neural network with a final softmax layer in the usual way, as if it were an image.
Consider a model which is to classify the sentence "Supreme Court to Consider Release of Mueller Grand Jury Materials to Congress" into one of two categories, 'politics' or 'sport'.
Each of the 12 words in the sentence is converted to a vector, and these vectors are joined together into a matrix. Here we are using a word vector size of 5 but in practice, large numbers such as 300 are often used.

This 2D matrix can be treated as an image and passed through a regular convolutional neural network, which outputs a probability for each class and which can be trained using backpropagation as in the 'cat' vs 'dog' example.
Because sentence lengths can vary, but the size of the input image to a network must be fixed, if a sentence is shorter than the maximum size then the unused values of the matrix can be padded with an appropriate value such as zeroes.

This approach to text classification also has the limitation that it cannot process sentences longer than the width of the input matrix. One workaround to this problem involves splitting sentences up into segments, passing each segment through the network individually, and averaging the output of the network over all sentences.

Convolutional Neural Networks for Drug Discovery

The first stage of a drug development program is drug discovery, where a pharmaceutical company identifies candidate compounds which are more likely to interact with the body in a certain way. Testing candidate molecules in pre-clinical or clinical trials is expensive, and so it is advantageous to be able to screen molecules as early as possible.

Proteins which play an important role in a disease are known as ‘targets’. There are targets that can cause inflammation or help tumors grow. The goal of drug discovery is to identify molecules that will interact with the target for a particular disease. The drug molecule must have the appropriate shape to interact with the target and bind to it, like a key fitting in a lock.

The San Francisco based startup Atomwise developed an algorithm called AtomNet, based on a convolutional neural network, which was able to analyze and predict interactions between molecules. Without being taught the rules of chemistry, AtomNet was able to learn essential organic chemical interactions. Atomwise was able to use AtomNet to identify lead candidates for drug research programs. AtomNet successfully identified a candidate treatment for the Ebola virus, which had previously not been known to have any antiviral activity. The molecule later went on to pre-clinical trials.

German Traffic Sign Recognition

Setup:

1. Exploratory data analysis
2. Data augmentation and preprocessing
3. Model architecture
4. Training
5. Model performance on German sign test data and model performance on unseen data

In [4]:
```
Annotation = namedtuple('Annotation', ['filename', 'label'])
def read_annotations(filename):
    annotations = []

    with open(filename) as f:
        reader = csv.reader(f, delimiter=';')
        next(reader) # skip header

        # loop over all images in current annotations file
        for row in reader:
            filename = row[0] # filename is in the 0th column
            label = int(row[7]) # label is in the 7th column
            annotations.append(Annotation(filename, label))

    return annotations
```

In [5]:
```
def load_training_annotations(source_path):
    annotations = []
    for c in range(0,43):
        filename = os.path.join(source_path, format(c, '05d'), 'GT-' + format(c, '05d') + '.csv')
        annotations.extend(read_annotations(filename))
    return annotations

def copy_files(label, filenames, source, destination, move=False):
    func = os.rename if move else shutil.copyfile

    label_path = os.path.join(destination, str(label))
    if not os.path.exists(label_path):
        os.makedirs(label_path)

    for filename in filenames:
        destination_path = os.path.join(label_path, filename)
        if not os.path.exists(destination_path):
            func(os.path.join(source, format(label, '05d'), filename), destination_path)

def split_train_validation_sets(source_path, train_path, validation_path, all_path, validation_fraction=0.2):
    """
    Splits the GTSRB training set into training and validation sets.
    """

    if not os.path.exists(train_path):
        os.makedirs(train_path)
```

In [1]:
```
# Put these at the top of every notebook, to get automatic reloading and inline plotting
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

In [2]:
```
# This file contains all the main external libs we'll use
from fastai.imports import *

from fastai.transforms import *
from fastai.conv_learner import *
from fastai.model import *
from fastai.dataset import *
from fastai.sgdr import *
from fastai.plots import *
from fastai.metrics import *

import matplotlib.pyplot as plt
import csv
from collections import defaultdict, namedtuple
import os
import shutil

import pandas as pd

from sklearn.metrics import confusion_matrix
```

In [ ]:
```
# Download and unpack the training set and the test set

! wget http://benchmark.ini.rub.de/Dataset/GTSRB_Final_Training_Images.zip -P data
! wget http://benchmark.ini.rub.de/Dataset/GTSRB_Final_Test_Images.zip -P data
! wget http://benchmark.ini.rub.de/Dataset/GTSRB_Final_Test_GT.zip -P data
! unzip data/GTSRB_Final_Training_Images.zip -d data
! unzip data/GTSRB_Final_Test_Images.zip -d data
! unzip data/GTSRB_Final_Test_GT.zip -d data

# Move the test set to data/test

! mkdir data/test
```

Exploratory analysis

In [8]:
```
classes = pd.read_csv('data/signnames.csv')
class_names = {}
for i, row in classes.iterrows():
    class_names[str(row[0])] = row[1]
```

In [183…
```
arch=resnet34
sz = 96
data = ImageClassifierData.from_paths(path, tfms=tfms_from_model(arch, sz), test_name='test')
```

```
augmentated training samples are shuffled
```

In [10]:
```
plt.imshow(load_img_id(data.val_ds, 1, path))
plt.show()
```

In [11]:
```
y = data.trn_ds.y
```

In [12]:
```
print(y[:10])
```

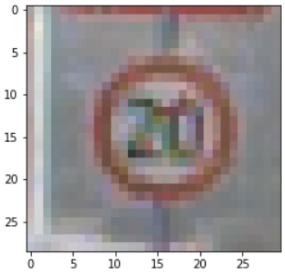
```
[0 0 0 0 0 0 0 0 0 0]
```

In [13]:
```
from collections import Counter

label_counts = Counter(y).most_common()
for l, c in label_counts:
    print(c, '\t', data.classes[l], '\t', class_names[data.classes[l]])
```

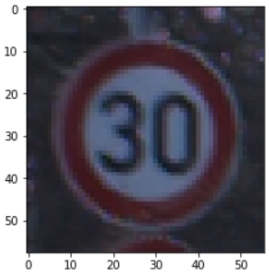
```
1800    1       Speed limit (30km/h)
1800    2       Speed limit (50km/h)
1740    13      Yield
1680    12      Priority road
1680    38      Keep right
1620    10      No passing for vechiles over 3.5 metric tons
1590    4       Speed limit (70km/h)
1500    5       Speed limit (80km/h)
1200    25      Road work
1200    9       No passing
1170    7       Speed limit (100km/h)
1140    3       Speed limit (60km/h)
1140    8       Speed limit (120km/h)
1080    11      Right-of-way at the next intersection
960     18      General caution
960     35      Ahead only
900     17      No entry
630     14      Stop
630     31      Wild animals crossing
569     33      Turn right ahead
510     15      No vechiles
480     26      Traffic signals
450     28      Children crossing
420     23      Slippery road
360     16      Vechiles over 3.5 metric tons prohibited
360     30      Beware of ice/snow
360     34      Turn left ahead
360     6       End of speed limit (80km/h)
330     22      Bumpy road
330     36      Go straight or right
300     20      Dangerous curve to the right
300     40      Roundabout mandatory
270     21      Double curve
240     24      Road narrows on the right
240     29      Bicycles crossing
240     39      Keep left
```

In [14]:
```
for label in sorted([l for l, c in label_counts], key=lambda p: int(data.classes[p])):
    i = [i for i, l in enumerate(y) if l == label][0]
    print(data.classes[y[i]], class_names[data.classes[y[i]]])
    plt.imshow(load_img_id(data.trn_ds, i, path))
    plt.show()
```

```
0 Speed limit (20km/h)
```



```
1 Speed limit (30km/h)
```



```
2 Speed limit (50km/h)
```

Image sizes

In [156…
```
flags = cv2.IMREAD_UNCHANGED+cv2.IMREAD_ANYDEPTH+cv2.IMREAD_ANYCOLOR

folder = 'data/all/0'

files = os.listdir(folder)
sizes = []

for file in files:
    filename = os.path.join(folder, file)
    img = cv2.imread(filename, flags)
    sizes.append(max(img.shape[0], img.shape[1]))
```

In [157…
```
plt.hist(sizes, bins=50)
plt.show()
```

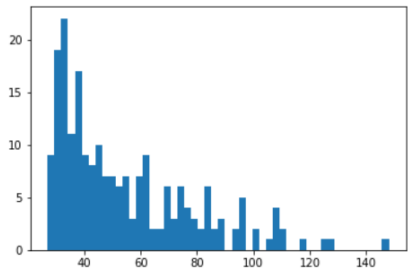
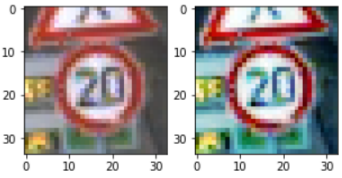


Image lighting

Let's try histogram equalization to improve constrast.

In [852…

```
flags = cv2.IMREAD_UNCHANGED+cv2.IMREAD_ANYDEPTH+cv2.IMREAD_ANYCOLOR

folder = 'data/all/0'

files = os.listdir(folder)
for i in range(5):
    f = plt.figure(figsize=(5, 5))

    filename = os.path.join(folder, files[i])
    img = cv2.imread(filename, flags)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    sp = f.add_subplot(1, 2, 1)
    plt.imshow(img)

    img = np.concatenate([np.expand_dims(cv2.equalizeHist(img[:,:,i]), axis=2) for i in range(3)], axis=2)


    sp = f.add_subplot(1, 2, 2)
    plt.imshow(img)
    plt.show()

    print(img.shape)
```



```
(34, 33, 3)
```


In [130…

```
# Normal version

def open_image_normal(fn):
    """ Opens an image using OpenCV given the file path.

    Arguments:
        fn: the file path of the image

    Returns:
        The numpy array representation of the image in the RGB format
    """
    flags = cv2.IMREAD_UNCHANGED+cv2.IMREAD_ANYDEPTH+cv2.IMREAD_ANYCOLOR
    if not os.path.exists(fn):
        raise OSError('No such file or directory: {}'.format(fn))
    elif os.path.isdir(fn):
        raise OSError('Is a directory: {}'.format(fn))
    else:
        try:
            return cv2.cvtColor(cv2.imread(fn, flags), cv2.COLOR_BGR2RGB).astype(np.float32)/255
        except Exception as e:
            raise OSError('Error handling image at: {}'.format(fn)) from e
```

In [123…

```
# Histogram equalization

def open_image_hist_eq(fn):
    """ Opens an image using OpenCV given the file path.

    Arguments:
        fn: the file path of the image

    Returns:
        The numpy array representation of the image in the RGB format
    """
    flags = cv2.IMREAD_UNCHANGED+cv2.IMREAD_ANYDEPTH+cv2.IMREAD_ANYCOLOR
    if not os.path.exists(fn):
        raise OSError('No such file or directory: {}'.format(fn))
    elif os.path.isdir(fn):
        raise OSError('Is a directory: {}'.format(fn))
    else:
        try:
            img = cv2.cvtColor(cv2.imread(fn, flags), cv2.COLOR_BGR2RGB)
```

Image augmentation

Here we can change image augmentation parameters and see how augmented images look like.

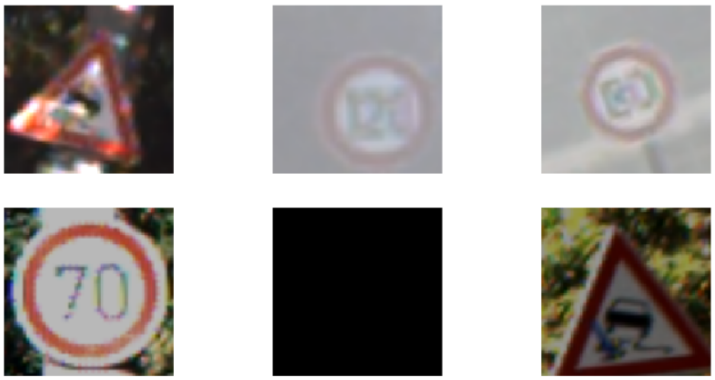
In [259…

```
# Look at examples of image augmentation
def get_augs():
    x,_ = next(iter(data.aug_dl))
    return data.trn_ds.denorm(x)[1]
```

In [259…

```
bs = 256

aug_tfms = [RandomRotate(20), RandomLighting(0.8, 0.8)]
tfms = tfms_from_model(arch, sz, aug_tfms=aug_tfms, max_zoom=1.2)
data = ImageClassifierData.from_paths(path, tfms=tfms, test_name='test', bs=bs)

ims = np.stack([get_augs() for i in range(6)])
plots(ims, rows=2)
```

```
augmentated training samples are shuffled
```



Training

In [259…

```
# Create a learner
learn = ConvLearner.pretrained(arch, data, precompute=False)
```

In [259…

```
wd=5e-4
```

Searching for a good starting learning rate

In [259…

```
def plot_loss_change(sched, sma=1, n_skip=20, y_lim=(-0.01,0.01)):
    """
    Plots rate of change of the loss function.
    Parameters:
        sched - learning rate scheduler, an instance of LR_Finder class.
        sma - number of batches for simple moving average to smooth out the curve.
        n_skip - number of batches to skip on the left.
        y_lim - limits for the y axis.
    """
    derivatives = [0] * (sma + 1)
    for i in range(1 + sma, len(learn.sched.lrs)):
        derivative = (learn.sched.losses[i] - learn.sched.losses[i - sma]) / sma
        derivatives.append(derivative)

    plt.ylabel("d/loss")
    plt.xlabel("learning rate (log scale)")
    plt.plot(learn.sched.lrs[n_skip:], derivatives[n_skip:])
    plt.xscale('log')
    plt.ylim(y_lim)
```

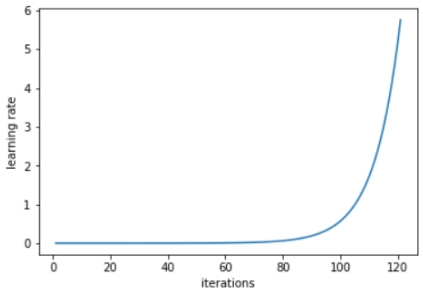
In [259…

```
learn.lr_find()
```

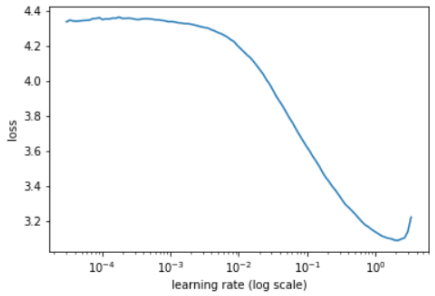
Failed to display Jupyter Widget of type `HBox`.

If you're reading this message in the Jupyter Notebook or JupyterLab Notebook, it may mean that the widgets JavaScript is still loading. If this message persists, it likely means that the widgets JavaScript library is either not installed or not enabled. See the Jupyter Widgets Documentation for setup instructions.

In [259…
```
learn.sched.plot_lr()
```



In [259…
```
learn.sched.plot()
```



In [259…
```
plot_loss_change(learn.sched, sma=20, n_skip=20, y_lim=(-0.02, 0.01))
```

Train for a few cycles

In [260…
```
learn.load('1')
```

In [261…
```
lr=0.01
```

In [261…
```
learn.fit(lr, 4, cycle_len=1, cycle_mult=2, wds=wd)
#learn.fit(lr, 4, cycle_len=1, cycle_mult=2)
```

Failed to display Jupyter Widget of type `HBox`.

If you're reading this message in the Jupyter Notebook or JupyterLab Notebook, it may mean that the widgets JavaScript is still loading. If this message persists, it likely means that the widgets JavaScript library is either not installed or not enabled. See the Jupyter Widgets Documentation for setup instructions.

If you're reading this message in another frontend (for example, a static rendering on GitHub or NBViewer), it may mean that your frontend doesn't currently support widgets.

```
 10%|▉         | 13/125 [00:05<00:47,  2.35it/s, loss=0.64]
Exception in thread Thread-1241:
Traceback (most recent call last):
  File "/opt/anaconda/envs/fastai/lib/python3.6/threading.py", line 916, in _bootstrap_inner
    self.run()
  File "/opt/anaconda/envs/fastai/lib/python3.6/site-packages/tqdm/_tqdm.py", line 144, in run
    for instance in self.tqdm_cls._instances:
  File "/opt/anaconda/envs/fastai/lib/python3.6/_weakrefset.py", line 60, in __iter__
    for itemref in self.data:
RuntimeError: Set changed size during iteration

epoch      trn_loss   val_loss   accuracy
    0      0.651346   0.047991   0.985439
    1      0.64426    0.035561   0.989346
    2      0.614354   0.028336   0.990288
    3      0.630812   0.048213   0.986517
    4      0.612389   0.034211   0.989494
    5      0.595143   0.032516   0.990167
    6      0.577719   0.033202   0.989763
    7      0.627798   0.049413   0.984779
```

Error analysis

In [265…
```
log_preds,y = learn.predict_with_targs()
preds = np.exp(log_preds)
pred_labels = np.argmax(preds, axis=1)
```

In [266…
```
cm = confusion_matrix(y, pred_labels)
```

In [266…
```
cm
```

Out[266…
```
array([[ 30,   0,   0, ...,   0,   0,   0],
       [  0, 420,   0, ...,   0,   0,   0],
       [  0,   0, 390, ...,   0,   0,   0],
       ...,
       [  0,   0,   0, ..., 270,   0,   0],
       [  0,   0,   0, ...,   0, 270,   0],
       [  0,   0,   0, ...,   0,   0, 269]])
```

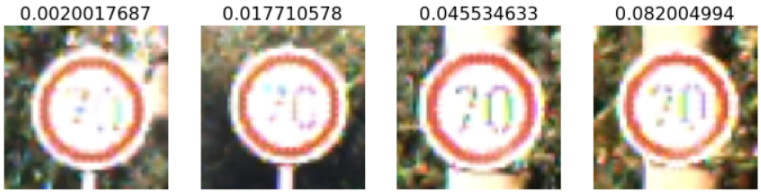
In [266…
```
results = ImageModelResults(data.val_ds, log_preds)
```

Most incorrect

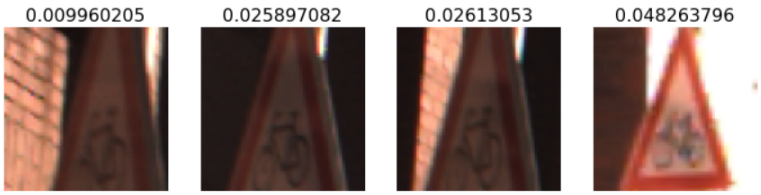
In [266…
```
incorrect = [i for i in range(len(pred_labels)) if pred_labels[i] != y[i]]
c = Counter([(y[i], class_names[data.classes[y[i]]]) for i in incorrect])
c.most_common(20)
```

Out[266…
```
[((11, 'Dangerous curve to the left'), 17),
 ((22, 'Bicycles crossing'), 16),
 ((7, 'No vechiles'), 12),
 ((34, 'Speed limit (70km/h)'), 7),
 ((19, 'Traffic signals'), 4),
 ((23, 'Speed limit (60km/h)'), 3),
 ((26, 'End of all speed and passing limits'), 3),
 ((29, 'Ahead only'), 3),
 ((17, 'Road narrows on the right'), 2),
 ((24, 'Beware of ice/snow'), 2),
 ((12, 'Speed limit (50km/h)'), 1),
```

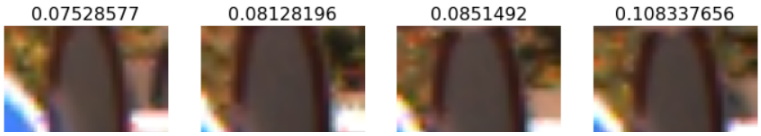
In [270…
```
results.plot_most_incorrect(34)
```



In [266…
```
results.plot_most_incorrect(22)
```



In [266…
```
results.plot_most_incorrect(7)
```

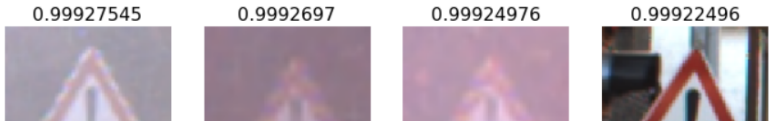


Most correct

In [267…
```
results.plot_most_correct(0)
```

In [267…
```
results.plot_most_correct(5)
```

In [267…
```
results.plot_most_correct(10)
```



Most uncertain

In [267…
```
results.plot_most_uncertain(1)
```

Test time augmentation (TTA)

In [263…
```
log_preds,y = learn.TTA(n_aug=20)
preds = np.mean(np.exp(log_preds),0)
```

In [263…
```
accuracy_np(preds, y)
```

Out[263…
```
0.9799725651577503
```

Retrain on the training set + validation set

Validation loss/accuracy won't be indicative of the model performance because the validation set is a subset of the training set now.

In [267…
```
arch = resnet34
sz = 96
bs = 256
wd = 5e-4
```

Testing on the test set

In [268…
```
true_test_labels = {a.filename: a.label for a in test_annotations}
class_indexes = {c: i for i, c in enumerate(data.classes)}
filenames = [filepath[filepath.find('/') + 1:] for filepath in data.test_ds.fnames]
labels = [str(true_test_labels[filename]) for filename in filenames]
y_true = np.array([class_indexes[label] for label in labels])
```

In [268…
```
log_preds = learn.predict(is_test=True)
preds = np.exp(log_preds)
accuracy_np(preds, y_true)
```

Out[268…
```
0.9929532858273951
```

In [268…
```
log_preds,_ = learn.TTA(n_aug=20, is_test=True)
preds = np.mean(np.exp(log_preds),0)
accuracy_np(preds, y_true)
```

Out[268…
```
0.9961203483768805
```

Analyze test results

In [268…
```
pred_labels = np.argmax(preds, axis=1)
```

In [269…
```
incorrect = [i for i in range(len(pred_labels)) if pred_labels[i] != y_true[i]]
```

In [269…
```
for i in range(0,10):
    print(class_names[data.classes[y_true[incorrect[i]]]], class_names[data.classes[pred_labels[incorrect[i]]]],
          preds[incorrect[i], y_true[incorrect[i]]], preds[incorrect[i], pred_labels[incorrect[i]]])
    plt.imshow(load_img_id(data.test_ds, incorrect[i], path))
    plt.show()
```

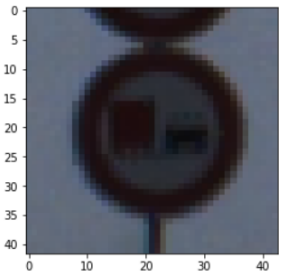
Create a submission file

In [269…
```
pred_labels = np.argmax(preds, axis=1)
pred_labels.shape
```

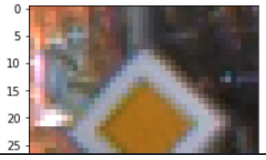
Out[269…
```
(12630,)
```

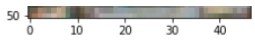
In [269…
```
for i in range(10):
    class_id = data.classes[pred_labels[i]]
    filename = data.test_ds.fnames[i].split('/')[1]
    print(filename, class_id, class_names[class_id])
    plt.imshow(load_img_id(data.test_ds, i, path))
    plt.show()
```

```
01154.ppm 10 No passing for vechiles over 3.5 metric tons
```

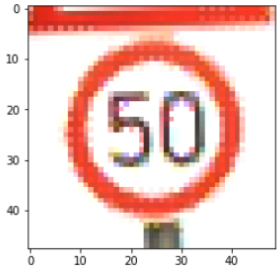


```
00920.ppm 12 Priority road
```

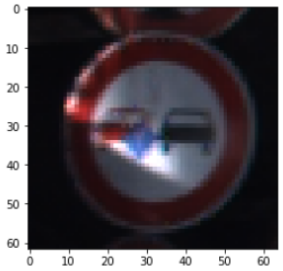




03602.ppm 2 Speed limit (50km/h)



12548.ppm 9 No passing



In [269…

```
with open('data/submission.csv', 'w') as f:
    for i in range(pred_labels.shape[0]):
        filename = data.test_ds.fnames[i].split('/')[1]
        f.write('{};{}\n'.format(filename, data.classes[pred_labels[i]]))
```

In [269…

```
from IPython.display import FileLink
FileLink('data/submission.csv')
```

Result:

The model achieved 99.22% accuracy on the validation set (random 20% subset of the training dataset).

REFERENCES

[1] https://benchmark.ini.rub.de/?section=home&subsection=news

[2] https://cg.cs.tsinghua.edu.cn/traffic-sign

[3] https://udai.gitbook.io/practical-ml/nn/training-and-debugging-of-nn

[4] https://www.tensorflow.org/guide/data

[5] https://arxiv.org/pdf/1506.01497.pdf