✦ Member-only story

Distracted Driver Detection using Deep Learning

Satya Naren Pachigolla · Follow
Published in Towards Data Science
10 min read · Dec 13, 2019

> *This blog and the project are a joint contribution of Amey Athaley, Apoorva Jasti, Sadhana Koneni, Satya Naren Pachigolla & Jayant Raisinghani under the guidance of Prof. Joydeep Ghosh*
> *Please follow this GitHub Repository for our implementation code*

Introduction

Driving a car is a complex task, and it requires complete attention. Distracted driving is any activity that takes away the driver's attention from the road. Several studies have identified three main types of distraction: visual distractions (driver's eyes off the road), manual distractions (driver's hands off the wheel) and cognitive distractions (driver's mind off the driving task).

The National Highway Traffic Safety Administration (NHTSA) reported that 36,750 people died in motor vehicle crashes in 2018, and 12% of it was due to distracted driving. Texting is the most alarming distraction. Sending or reading a text takes your eyes off the road for 5 seconds. At 55 mph, that's like driving the length of an entire football field with your eyes closed.

Many states now have laws against texting, talking on a cell phone, and other distractions while driving. We believe that computer vision can augment the efforts of the governments to prevent accidents caused by distracted driving. Our algorithm automatically detects the distracted activity of the drivers and alerts them. We envision this type of product being embedded in cars to prevent accidents due to distracted driving.

Data

We took the StateFarm dataset which contained snapshots from a video captured by a camera mounted in the car. The training set has **~22.4 K** labeled samples with equal distribution among the classes and **79.7 K** unlabeled test samples. There are 10 classes of images:

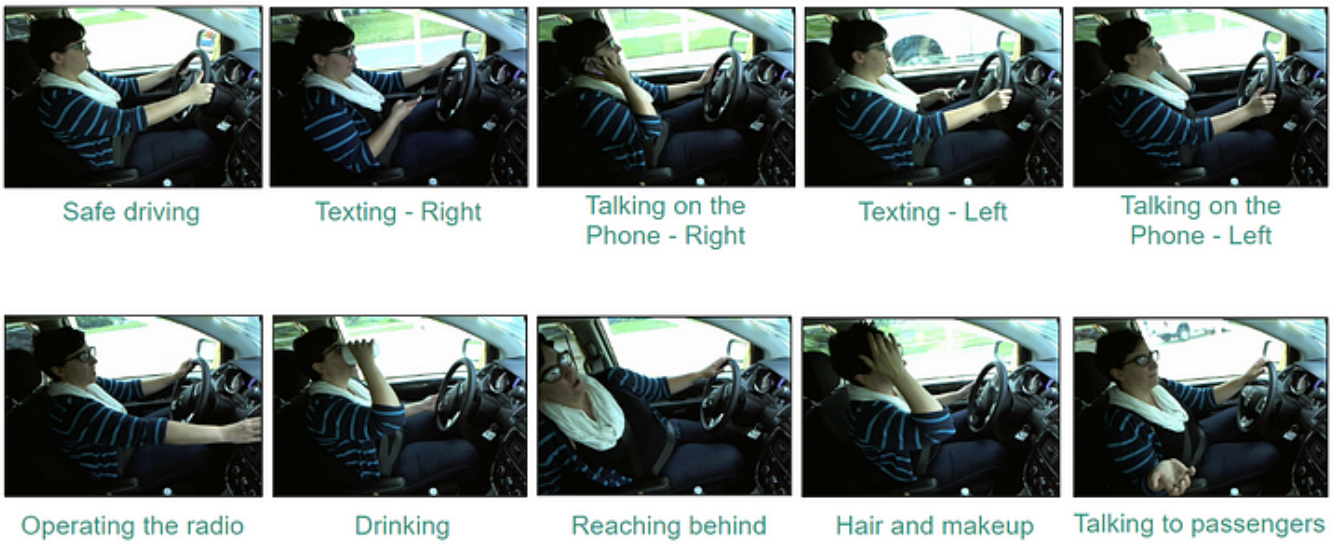


Fig: Sample images from the training data

Evaluation Metric

Before proceeding to build models, it's important to choose the right metric to gauge its performance. Accuracy is the first metric that comes to mind. But, accuracy is not the best metric for classification problems. Accuracy only takes into account the correctness of the prediction i.e. whether the predicted label is the same as the true label. But, the confidence with which we classify a driver's action as distracted is very important in evaluating the performance of the model. Thankfully, we have a metric that captures just that — **Log Loss**.

Logarithmic loss (related to cross-entropy) measures the performance of a classification model where the prediction input is a probability value between 0 and 1. The goal of our machine learning models is to minimize this value. A perfect model would have a log loss of 0 and it increases as the predicted probability diverges from the actual label. So predicting a probability of 0.3 when the actual observation label is 1 would result in a high log loss

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$$

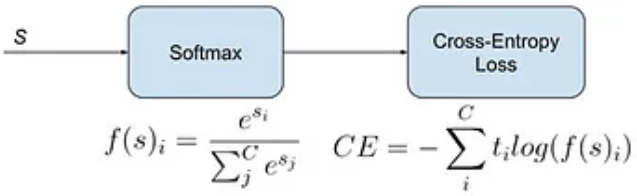


Fig: Evaluation Metric

Data leakage

With the understanding of what needs to be achieved, we proceeded to build the CNN models from scratch. We added the usual suspects — convolution batch normalization, max pooling, and dense layers. The results — loss of 0.014 and accuracy of 99.6% on the validation set in 3 epochs.

Epochs: 3	Loss	Accuracy
Train	0.0034	99.9%
Validation	0.014	99.6%

Fig: Initial Model Results

Well, we contemplated for a second about accidentally building the best CNN architecture the world has ever seen. So, we predicted the classes for the unlabeled test set using this model.

The Moment of Truth

Fig: Class prediction by the model

Oh, well. There was no serendipity after all. So, we looked deeper into what could have gone wrong and we found that our training data has multiple images of the same person within a class with slight changes of angle and/or shifts in height or width. This was causing a data leakage problem as the similar images were in validation as well, i.e. the model was trained much of the same information that it was trying to predict.

Solution to Data Leakage

To counter the issue of data leakage, we split the images based on the person IDs instead of using a random 80–20 split.

Now, we see more realistic results when we fit our model with the modified training and validation sets. We achieved a loss of 1.76 and an accuracy of 38.5%.

Epochs: 15	Loss	Accuracy
Train	2.08	23.8%
Validation	1.76	38.5%

Fig: Model fit after countering data leakage

To improve the results further, we explored using the tried and tested deep neural nets architectures.

Transfer Learning

Transfer learning is a method where a model developed for a related task is reused as the starting point for a model on a second task. We can re-use the model weights from pre-trained models that were developed for standard computer vision benchmark datasets, such as the ImageNet image recognition challenge. Generally, the final layer with softmax activation is replaced to suit the number of classes in our dataset. In most cases, extra layers are also added to tailor the solution to the specific task.

It is a popular approach in deep learning considering the vast compute and time resources required to develop neural network models for image classification. Moreover, these models are usually trained on millions of images which helps especially when your training set is small. Most of these model architectures are proven winners — VGG16, RESNET50, Xception and Mobilenet models that we leveraged gave exceptional results on the ImageNet challenge.

Image Augmentation

```
datagen = ImageDataGenerator(
    height_shift_range=0.5,
    width_shift_range = 0.5,
    zoom_range = 0.5,
    rotation_range=30
        )
#datagen.fit(X_train)
data_generator = datagen.flow(X_train, y_train, batch_size = 64)
```

Fig: Sample code for Image Augmentation

Since our training image set had only ~22K images, we wanted to synthetically get more images from the training set to make sure the models don't overfit as the neural nets have millions of parameters. Image Augmentation is a technique that creates more images from the original by performing actions such as shifting width and/or height, rotation, and zoom. Refer to this article to know more about Image Augmentation.

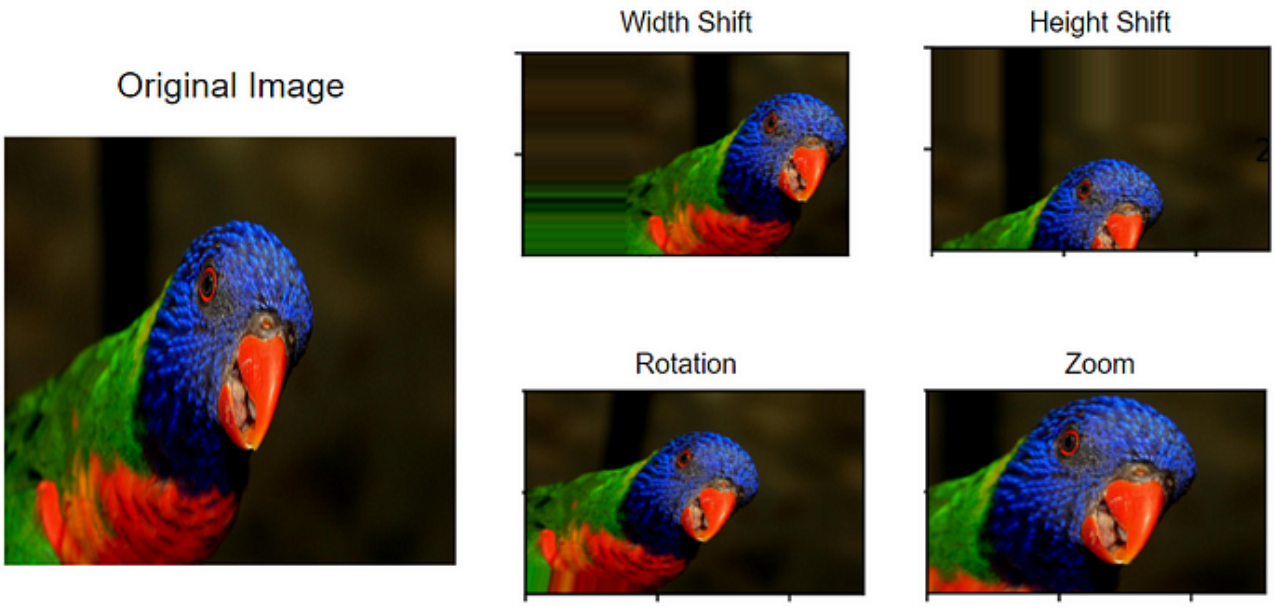


Fig: Types of image augmentation implemented in our dataset

For our project, Image Augmentation had a few additional advantages. Sometimes, the difference between images from the two different classes can be very subtle. In such cases, getting multiple looks at the same image through different angles will help. If you look at the images below, we see that they are almost similar but in the first picture the class is 'Talking on the Phone — Right' and the second picture belongs to the 'Hair and Makeup' class.

Fig: Sample of image class confusion (i) Taking on the phone (ii) Hair and Makeup

Extra Layers

To maximize the value from transfer learning, we added a few extra layers to help the model adapt to our use case. Purpose of each layer:

- **Global average pooling** layer retains only the average of the values in each patch
- **Dropout** layers help in controlling for overfitting as it drops a faction of parameters(bonus tip: it's a good idea to experiment with different dropout values)
- **Batch normalization** layer normalizes the inputs to the next layer which allows faster and more resilient training
- **Dense** layer is the regular fully-connected layer with a specific activation function

Which Layers to Train?

The first question when doing transfer learning is if we should train only the extra layers added to the pre-existing architecture or if we should train all the layers. Naturally, we started by using the ImageNet weights and trained only the new layers since the number of parameters to train would be lesser and the model would train faster. We saw that the accuracy on validation set plateaued at **70%** after 25 epochs. But, we were able to get an accuracy of **80%** by training all the layers. Hence, we decided to go ahead with training all the layers.

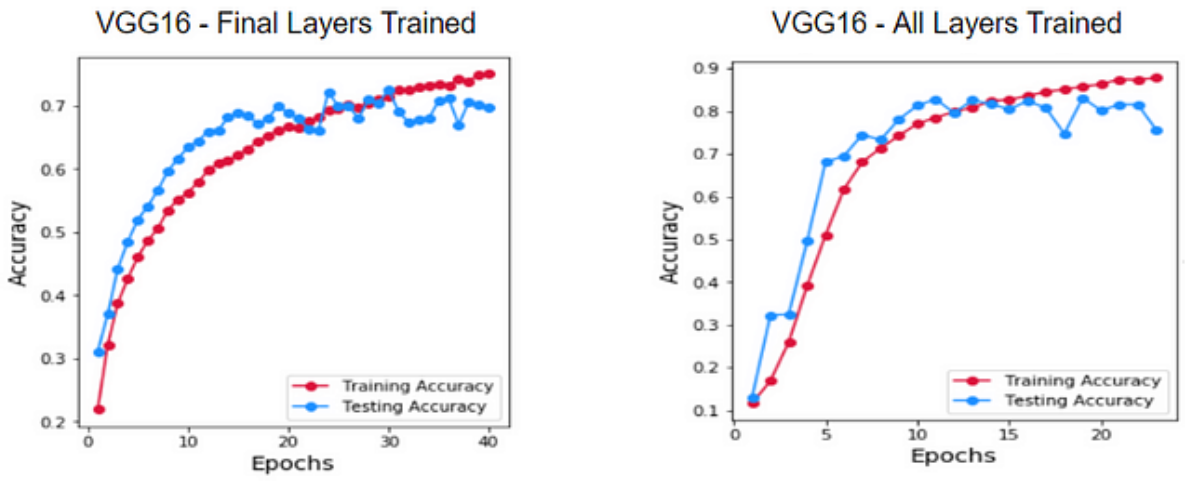


Fig: Comparison of model accuracy for final vs all trained layers

Which Optimizer to Use?

Optimizers minimize an objective function parameterized by a model's parameters by updating the parameters in the opposite direction of the gradient of the objective function w.r.t. to the parameters. To know more about how different optimizers work, you can refer to this blog.

The most popular algorithm in the deep learning world is Adam which combines SGD and RMS Prop. It has been consistently performing better than other optimizers for *most* problems. However, in our case, Adam showed erratic pattern of descent while SGD was learning gradually. By doing some literature survey, we found that in few cases SGD is superior to Adam because SGD generalizes better (link). As SGD was giving stable results, we used it for all our models.

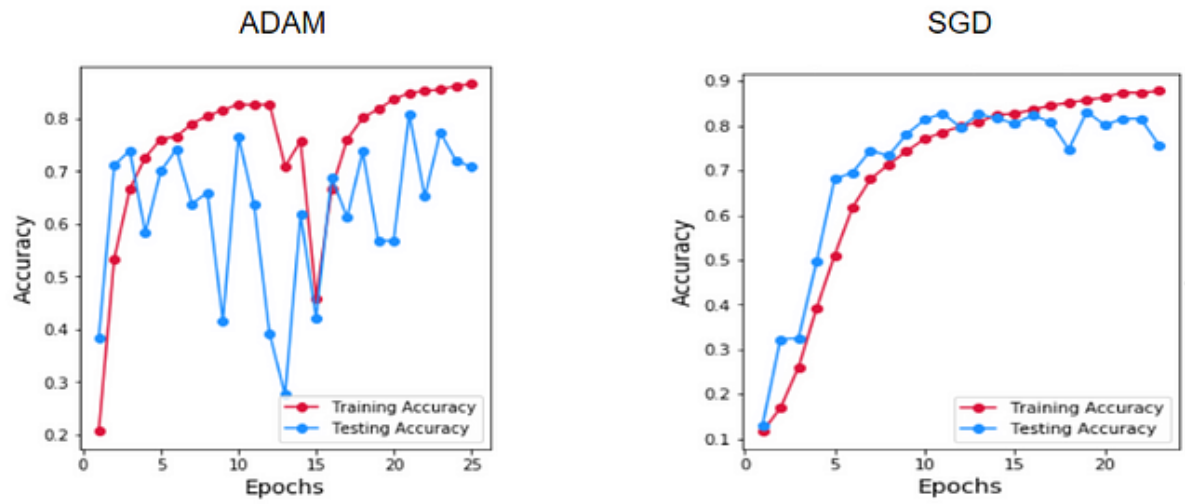


Fig: Accuracy across epochs using: (i)Adam (ii)SGD

Which Architectures to Use?

We tried multiple transfer learning models with the weights from training on the ImageNet dataset i.e. pre-trained weights.

- **VGG16**
  VGG16 model has 16-layers. It mainly uses convolutional techniques along with zero-padding, dropout, max-pooling and flattening.

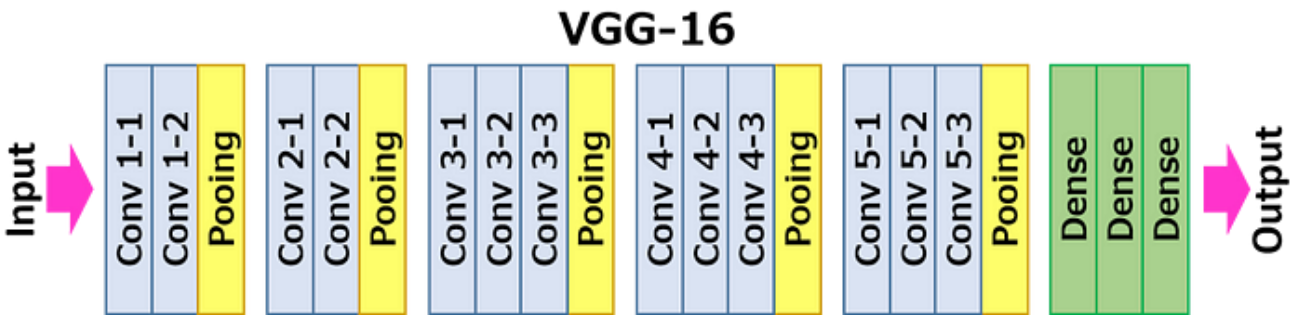


Fig: VGG-16 Architecture

- **RESNET50**
  RESNET50 is an extension of VGG16 model with 50 layers. To counter the issue of difficulty in training a deeper network, feedforward neural networks with "shortcut connections" with reference to the layer inputs have been introduced.

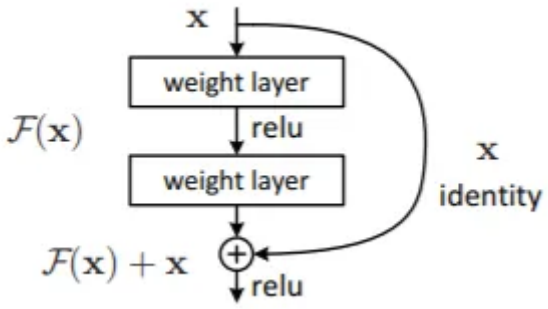


Fig: Residual learning: a building block

- <u>Xception</u>
  While RESNET was created with the intention of getting ***deeper*** networks, Xception was created for getting ***wider*** networks by introducing ***depthwise separable convolutions***. By decomposing a standard convolution layer into depthwise and pointwise convolutions, the number of computations reduces significantly. The performance of the model also improves because of having multiple filters looking at the same level.

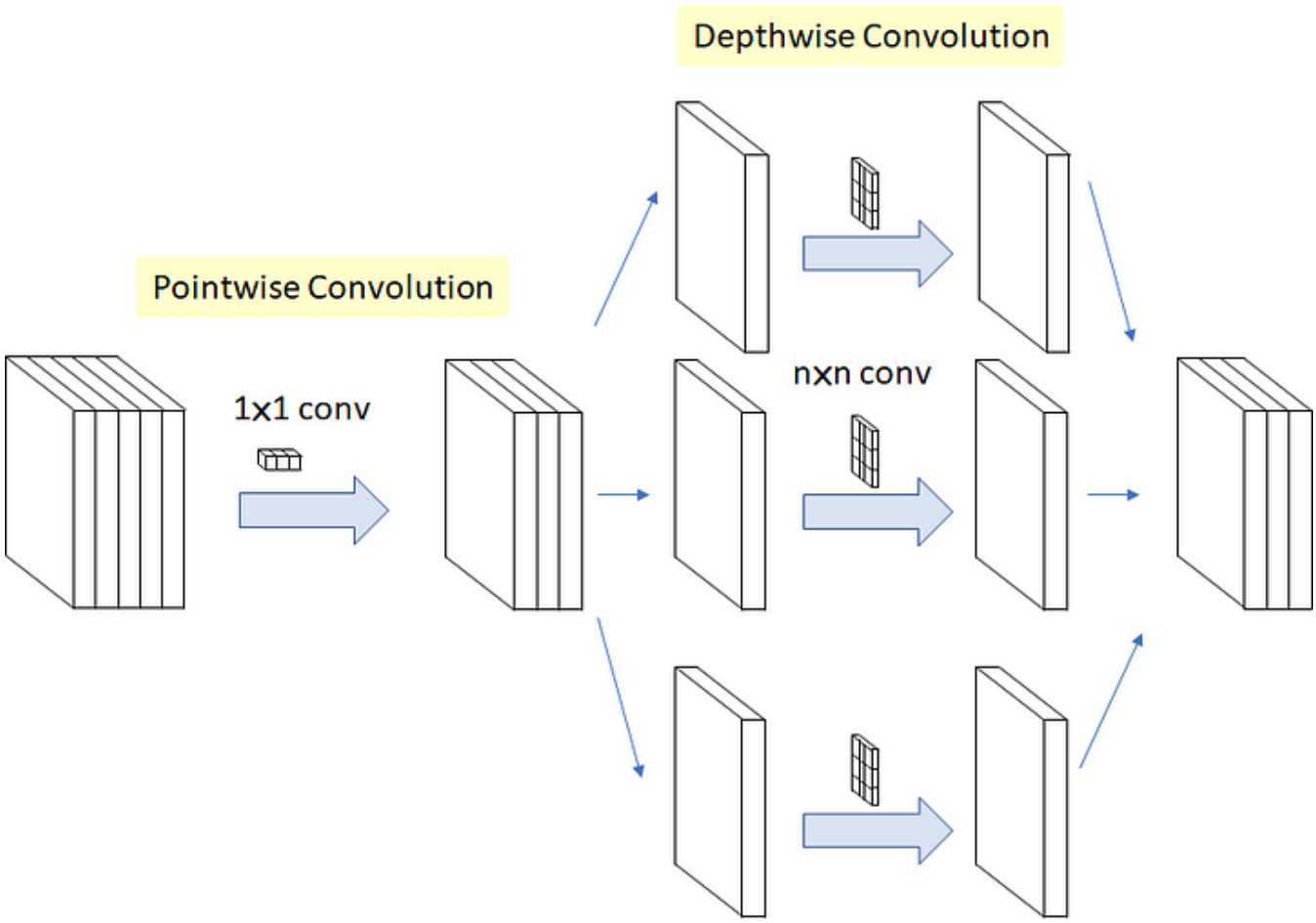


Fig: Modified Depthwise Separable Convolution used in Xception

- **MobileNet**
  MobileNet was a model developed by Google for ***mobile based vision applications***. It was proven to reduce computational costs by at least 9 times. MobileNet uses depth-wise separable convolutions to build light weight deep neural networks. It has two simple global hyper-parameters that efficiently trade off between latency and accuracy.

Performance of the Transfer Learning Models

	Extra Layers Added?	Train Loss	Validation Loss	Test Loss
VGG 16	No	0.41	0.5	0.64
VGG 16	Yes	0.42	0.57	0.58
RESNET 50	No	0.34	0.55	0.57
RESNET 50	Yes	0.32	0.47	0.45
Xception	No	0.4	0.55	0.57
Xception	Yes	0.42	0.52	0.45
Mobilenet	No	0.28	0.4	0.39

Fig: Comparison of Transfer Learning Models. MobileNet has the minimum loss on the test set

Comparing the Best Models

While each of the architectures above gave us good results, there is a significant variance in the performance of each model for individual classes. From the table below, we notice that different models have the best accuracy for each class. Hence, we decided to build an ensemble of these models.

	Xception	Resnet 50	VGG 16	Mobilenet
Safe Driving	78.9%	92.5%	54.4%	78.2%
Texting -Right	93.2%	92.3%	96.2%	89.1%
Talking - Right	99.4%	94.3%	95.1%	97.4%
Texting - Left	98.2%	98.2%	98.5%	98.5%
Talking - Left	76.5%	75.8%	87.9%	77.3%
Operating the Radio	94.8%	99.0%	96.8%	97.5%
Drinking	80.4%	86.2%	90.3%	79.3%
Reaching Behind	100.0%	99.4%	81.1%	99.7%
Hair and Makeup	40.1%	53.3%	63.6%	59.6%
Talking to passenger	58.8%	68.5%	83.1%	79.5%

Fig: Accuracy of different algorithms per class

Figure: Accuracy of each model for a specific class. ‘Green’ and ‘red’ indicate higher to lower accuracy

Ensemble Models

Now that we have 7 best models with high variance among the posterior probabilities, we tried multiple ensembling techniques to improve the log loss further.

- **Mean Ensembling:** This is the easiest and most widely used ensembling method where the posterior probability is calculated as the mean of the predicted probabilities from the component models.
- **Trimmed Mean Ensembling:** This is Mean Ensembling by excluding the maximum and minimum probabilities from the component models for each image. It helps in further smoothing our predictions leading to a lower log loss value.
- **KNN for Ensembling:** Since the images are all snapped from video snippets while drivers were engaged in a distracting activity or were driving, there are a substantial number of images from the same class that are similar. Based on this premise, finding similar images and averaging the probabilities over these images helped us smoothen predicted probabilities for each class.
  To find the 10 nearest neighbors, we used outputs from the penultimate layer of VGG16 transfer learning model as features on the validation set.

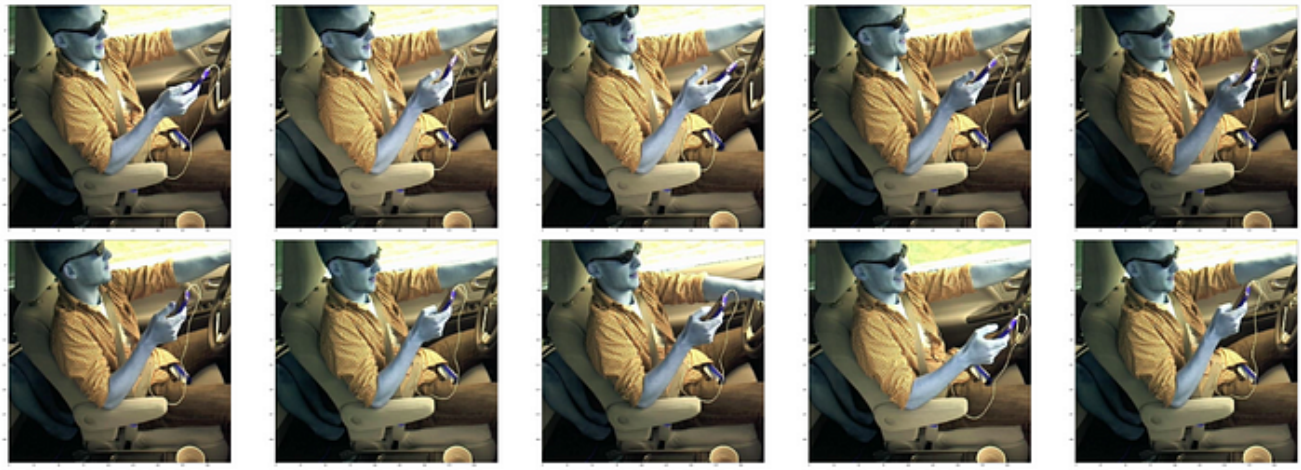


Fig: Output of KNN — 10 Nearest Neighbours

Model	Validation Loss	Validation Accuracy
MobileNet	0.4	85.7%
Mean Ensemble	0.32	89.7%
Trimmed Mean Ensemble	0.3	89.7%
KNN Ensemble	0.24	92.2%

Fig: Comparison of ensemble models with MobileNet model as the benchmark

Learnings

We believe that these learnings from our experience would benefit anyone who is working on a deep learning project for the first time like us:

1. Use Pickle Files: One free resource you can use for your project is **'Google Colab'**. You get access to the GPU which helps when working with huge data due to parallel computing. When using Colab, you can perform the necessary pre-processing steps by reading all your images once and save it in a pickle file. This way, you can pick up where you left off by directly loading the pickle files. You can then start training your models

2. Early Stopping and Call Backs: Generally, deep learning models are trained with a large number of epochs. In this process, the model might improve accuracy up to a few epochs and then starts diverging. The final weights stored at the end of training will not be the best values i.e. they might not give the minimum log loss. We can use the ***CallBacks*** feature in Keras which saves the weights of a model only if it sees improvement after an epoch. You can reduce the training time by using ***Early Stopping***, you can set a threshold on the number of epochs to run after the model stops seeing any improvement.

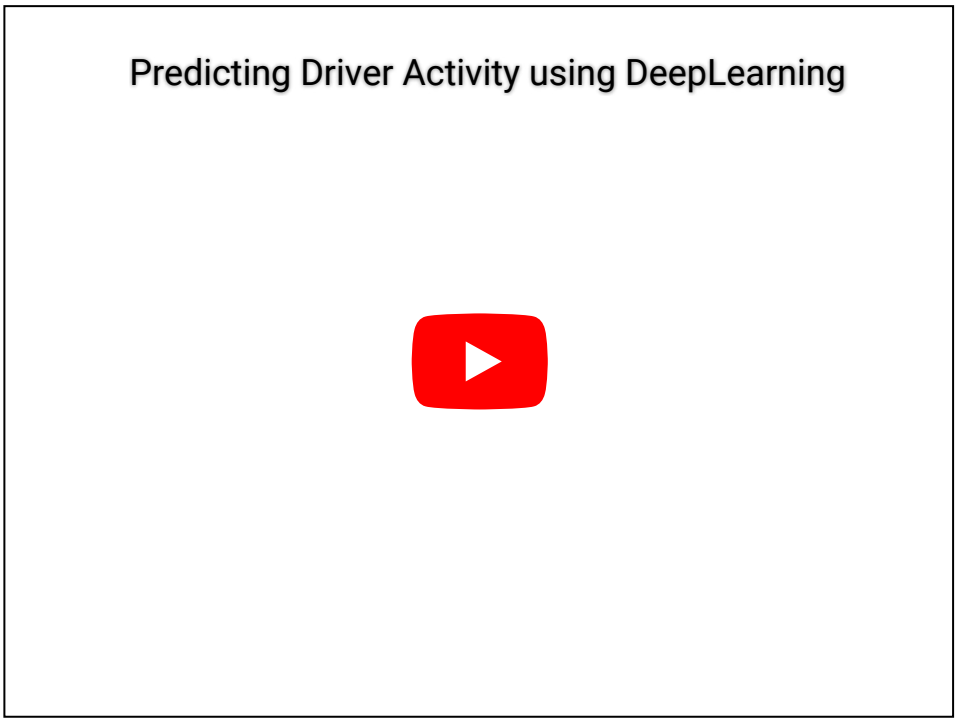
```
1  checkpointer = ModelCheckpoint('vgg_weights_aug_setval_layers_sgd2.hdf5', verbose=1, sav
2  earlystopper = EarlyStopping(monitor='val_loss', patience=10, verbose=1)
3
4  # These need to be passed as arguments for the fit_generator function
5
6  vgg16_model = vgg16_pretrained.fit_generator(data_generator,steps_per_epoch = len(X_trai
7                                                epochs = 25, verbose = 1, va
```

3. Mean or Trimmed Mean is better than Model Stacking for Ensembles: The inputs to your stacked model would have high correlations which causes the output to have high variance. Hence, in this case, the simpler approach is the best approach.

4. Never Lose Sight of the End Application: Doing an ensemble of 7 models and then KNN on the output gave us a good score but if we had to choose a single model that can be used to get good but faster predictions with least amount of resources, Mobilenet would be the obvious choice. Mobilenet is specifically developed with computational restraints in mind which suit the application in the car the best and it had the lowest log loss among the 7 stand-alone models

We believe that a device with a camera installed in cars that tracks the movements of the driver and alerts them can help prevent accidents.

To drive this point home, we created a small video which demonstrates how our model can be used:



Sample video depicting predictions

References:

1. Stanford CS231N lecture series for CNN : *https://www.youtube.com/watch?v=vT1JzLTH4G4&list=PLzUTmXVwsnXod6WNdg57Yc3zFx_f-RYsq&index=1*

2. How to implement CNN using Keras, TensorFlow: *https://www.youtube.com/watch?v=wQ8BIBpya2k&list=PLQVvvaa0QuDfhTox0AjmQ6tvTgMBZBEXN*

3. CNN Architectures: *https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5*

4. Transfer learning articles:

- *https://medium.com/r/?url=https%3A%2F%2Ftowardsdatascience.com%2Ftransfer-learning-using-mobilenet-and-keras-c75daf7ff299*
- *https://github.com/bdutta19/kaggle_statefarm*
- http://cs229.stanford.edu/proj2016/report/SamCenLuo-ClassificationOfDriverDistraction-report.pdf
- *https://arxiv.org/abs/1704.04861*

Machine Learning Computer Vision Transfer Learning Cnn

Image Classification

Follow

Written by Satya Naren Pachigolla

26 Followers · Writer for Towards Data Science

More from Satya Naren Pachigolla and Towards Data Science

Leonie Monigatti in Towards Data Science

Getting Started with LangChain: A Beginner's Guide to Building LLM-Powered Applications

A LangChain tutorial to build anything with large language models in Python

✦ · 12 min read · Apr 25

2.2K 18

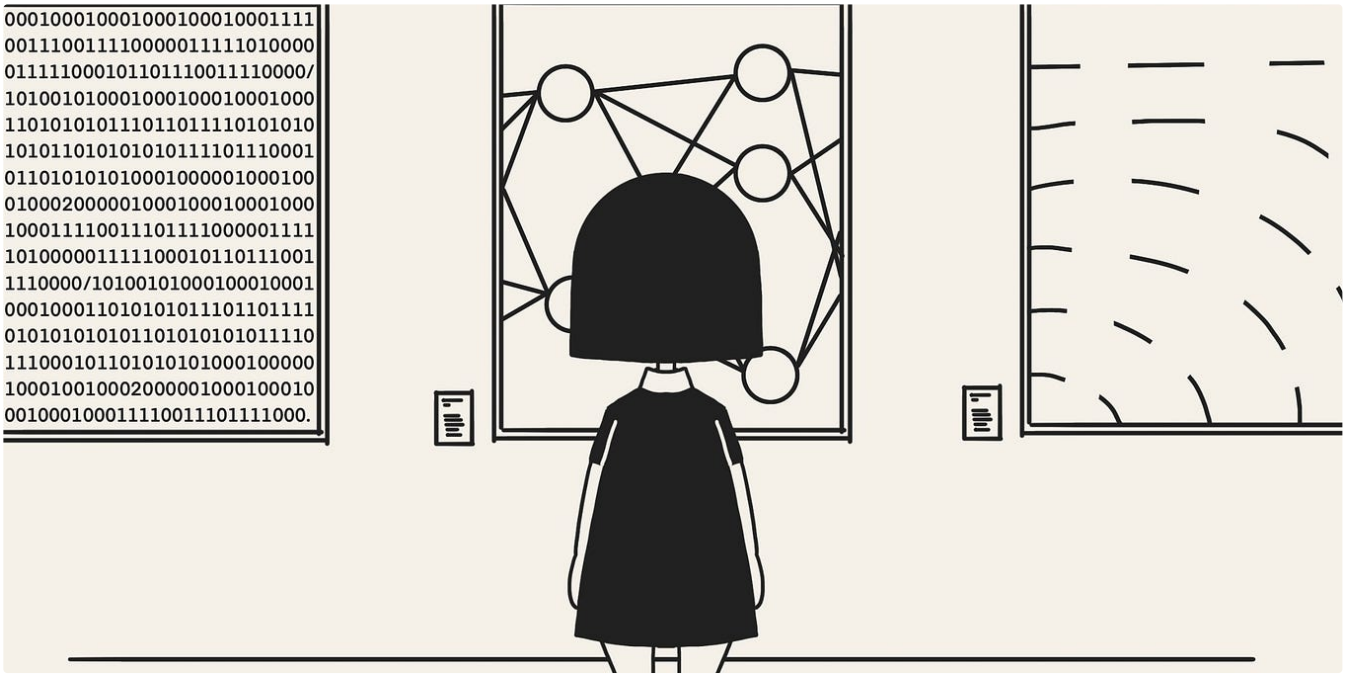
Matt Chapman in Towards Data Science

How I Stay Up to Date With the Latest AI Trends as a Full-Time Data Scientist

No, I don't just ask ChatGPT to tell me

· 8 min read · May 1

1.4K 22



Leonie Monigatti in Towards Data Science

10 Exciting Project Ideas Using Large Language Models (LLMs) for Your Portfolio

Learn how to build apps and showcase your skills with large language models (LLMs). Get started today!

✦ · 11 min read · May 15

816 5

Cassie Kozyrkov in Towards Data Science

The Best Learning Paths for AI and Data Leadership

How to muscle up on data-related topics quickly

✦ · 7 min read · Apr 30

1.4K 9

See all from Satya Naren Pachigolla

See all from Towards Data Science

Recommended from Medium

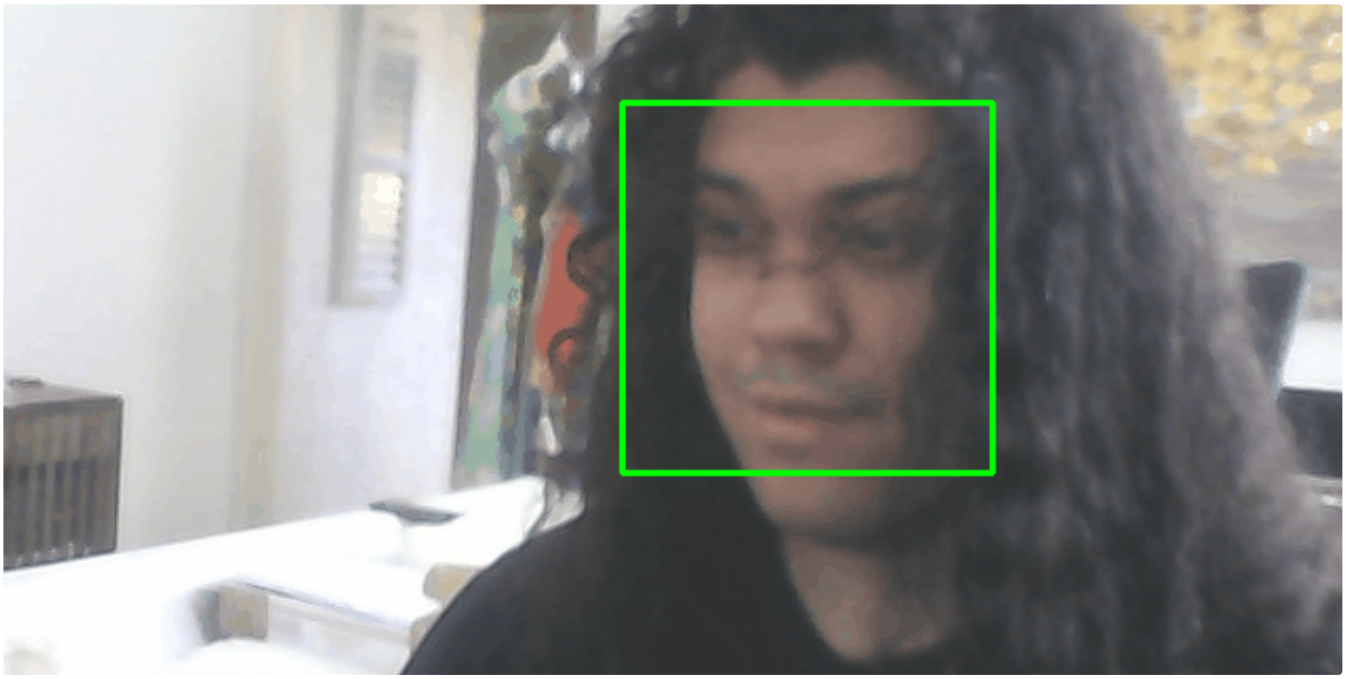
Bert Gollnick in MLearning.ai

Create a Custom Object Detection Model with YOLOv7

Train a model to detect face masks in real-time with the most powerful real-time algorithm YOLOv7

· 5 min read · Dec 10, 2022

101 1



Victor Murcia

Real-Time Facial Recognition with Python

One of the first computer vision projects I remember being really stoked about was object detection. More specifically, face detection. The...

· 4 min read · Jan 9

Lists

What is ChatGPT?

9 stories · 60 saves

Staff Picks

323 stories · 82 saves

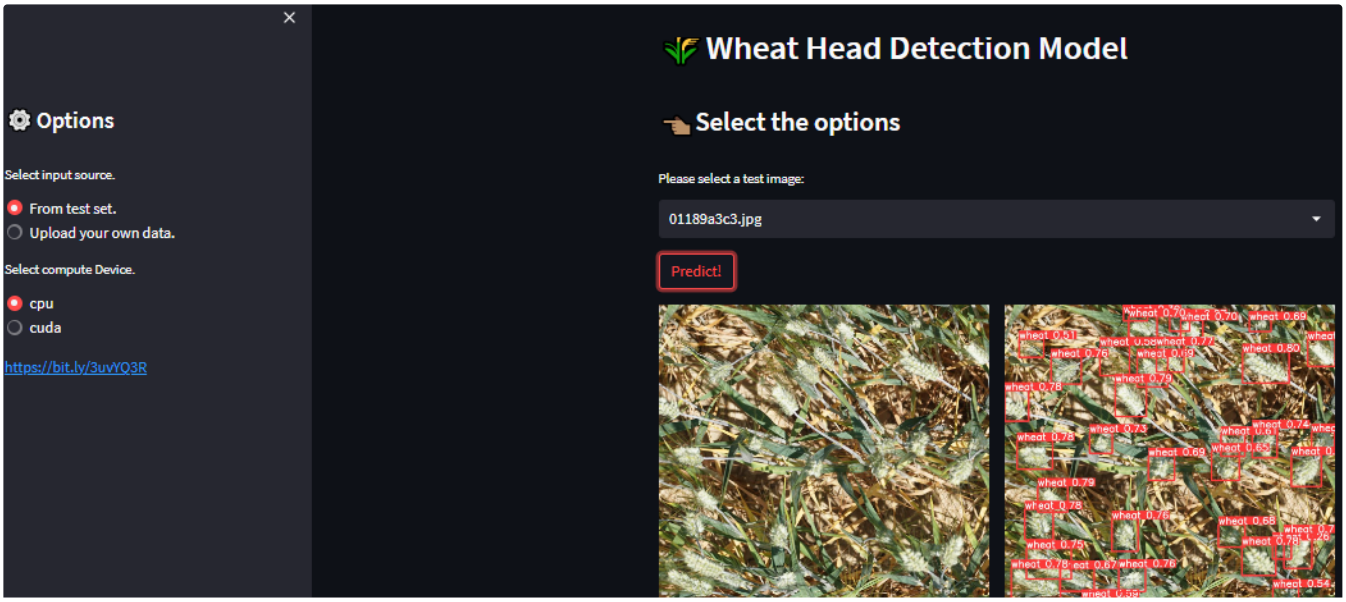
Matt Chapman in Towards Data Science

The Portfolio that Got Me a Data Scientist Job

Spoiler alert: It was surprisingly easy (and free) to make

✨ · 10 min read · Mar 24

2.9K 44



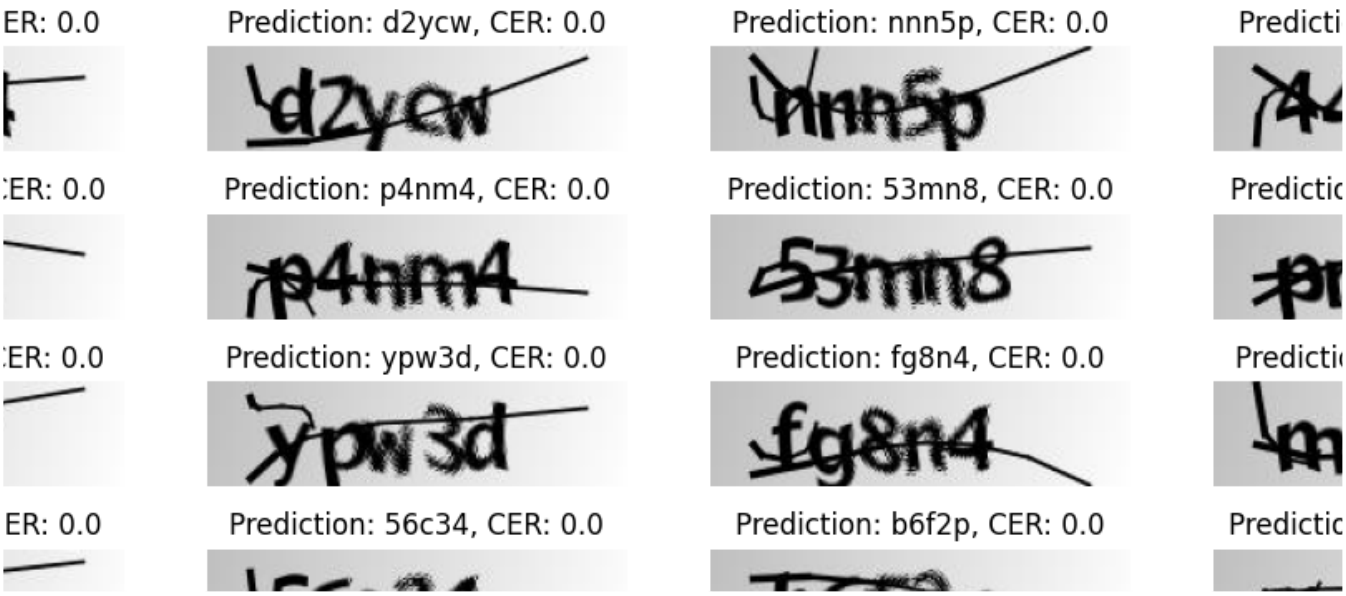
Ebrahim Haque Bhatti

YOLOv5 Tutorial on Custom Object Detection Using Kaggle Competition Dataset

And Deploying Using Streamlit

✦ · 8 min read · Dec 8, 2022

152



Rokas Liuberskis in Towards AI

TensorFlow OCR Model for Reading Captchas

Training a Custom OCR for Captcha Image Text Extraction with TensorFlow and CTC Loss Function: A Step-by-Step Guide. Using the mltu Library

✦ · 9 min read · Dec 23, 2022

141 1

Maryna Klokova in MLearning.ai

ML interview prep: Computer vision popular topics

Computer vision: tasks, libraries, networks, datasets, preprocessing, image transformations, augmentation, pipeline, metrics, edges and...

· 7 min read · Nov 29, 2022

193 1