Emotion Recognition Solution

Sergey Tarasenko, PhD

Table of Content

Task Definition

Why Emotion Recognition is important?

Human emotions play important role in human social life. Therefore, human emotion recognition is becoming increasingly vital in various industries as it:

- enhances customer interactions
- provide deep insight into current state of mind of a person
- improves employee engagement
- drives better decision-making.

By understanding emotional cues:

- businesses can tailor their services to meet client needs more effectively
- fostering loyalty
- increase client satisfaction

In the workplace, recognizing employee emotions can lead to a more supportive environment, promoting well-being and productivity.

Therefore, recognizing human emotions is a fundamental task to be solves. Since emotion are everywhere in our life, such a solution should be of high demand.

Task Definition

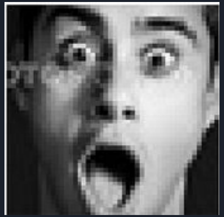
We are given a dataset of grayscale images, which contain human faces with 4 types of emotion expressions:

- Happy
- Neural
- Sad
- Surprise

Happy

Neutral

Sad


Surprise

Our task includes 3 key points:

- Develop a robust solution for Emotion recognition
- Explain system deployment and limitations
- Explain business value

Data Overview (EDA)

Data overview (EDA)

Provided images is already splitted into train, validation and test datasets. Each dataset constrain 4 classes, i.e., happy, neural, sad, surprise.

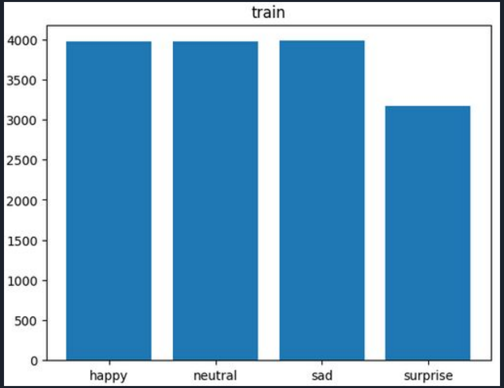
Key finding about the datasets:

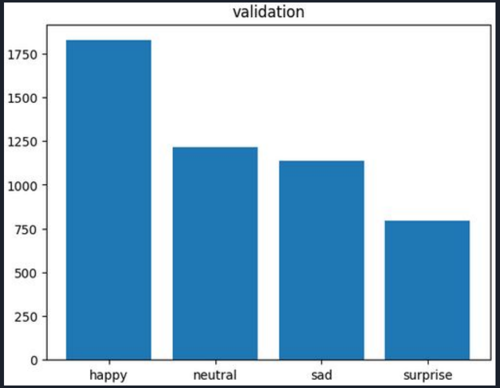
- Images in all three datasets contain faces of people of various genders (men, women), ages (child, teenager, adult), ethnicity
- Pictures of faces are taken from different distance, from different angles, against various backgrounds, face can appear in different location of an image
- Some images contain noise in a form of irrelevant text
- Some people wear glasses (both sunglasses and optical devices) with both transparent and completely opaque lens

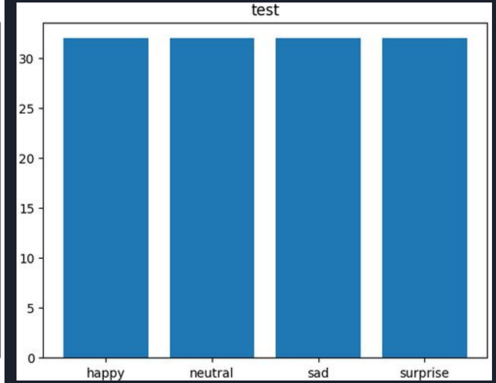
Data overview (EDA)

Key finding about the datasets:

- Training dataset contains 15109 images
- Validation dataset contains 4977 images
- Test dataset contains 128 images
- Images are distributed across classes for each data set as follows:







We can infer that in the case of:

- Train dataset: surprise class contains about 21% sample less than other 3 classes
- Test dataset : all four classes contain the same number of samples (32)
- Validation dataset has highly unbalanced distribution of images across classes

Model Development

Model Development

Model Type

Given the observations from EDA, we decided to stick to Convolutional Neural Networks as they are contain filters to account for spatial dependencies, shift-invariant recognition and trainable parameters' sharing.

Quality Metrics

Emotion recognition is a classification task. Therefore we have selected the following metrics:

- training , validation accuracy to trace model training
- Test accuracy to access model using test dataset
- Recall, precision and f1-sore for each class as well as their weighted average, to assess how well network is recognising classes
- Confusion matrix to visualize distribution of classified samples across classes. This also help to see the errors in form of False Positive and False Negative.

Model Development

Techniques to find a good model

- **Check RGB vs. GrayScale input**: RGB input models show higher performance
- **Under/Over sample Training Dataset:** Oversampling training dataset has a positive effect
- **Vary batch size**: model trained using batch size of 32 has the best performance
- **Vary kernel size**: model trained using kernel size 2x2 has the better performance when using kernel size 3x3
- **Add regularization** to eliminate overfitting: did not work in a good way, model performance dropped
- **Data augmentation**: did not provide any performance improvement
- **Use different CNN models** of different depth
- **Build CNN by increasing or decreasing number of filters** in each following layer: increasing number of filters had obvious positive effect
- **Apply transfer learning** using partial training (only classifier) or complete re-training: complete retraining has improvement effect for VGG16 model, but not for other models
- **Appy skip connections**: resulted in essential improvement
- Early stopping
- Learning rate decay

Model Development

Framework to manage experiments

Having various dimensions which influence model performance, requires effective mechanism of running multiple experiments and tracking and saving results.

For this purpose a unified frame to run experiments has been created.

Key features of the unified framework:

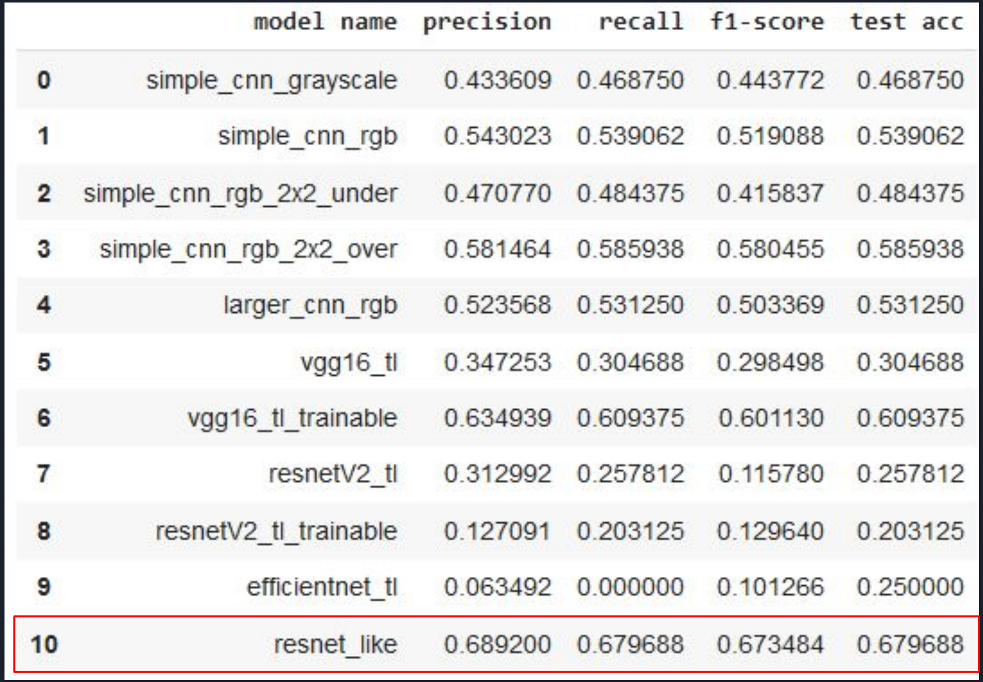
- Functions to compute predictions, confusion matrices and classification report
- Functions to save:
  - best models,
  - training and validation learning curves ,
  - confusion matrices
  - classification reports
  - Model configs
- Unified config format for models
- Collect results for all models in one dataframe

Model Development

Metrics for Selected Models:

- Simple CNN Grayscale
- Simple CNN RGB
- Simple CNN RGB kernel 2x2 trained on undersampled data
- Simple CNN RGB kernel 2x2 trained on oversampled data
- Larger CNN RGB
- VGG16 Transfer Learning non-trainable
- VGG16 Transfer Learning trainable
- ResNetV2 Transfer Learning non-trainable
- ResNetV2 Transfer Learning trainable
- Efficient Net Transfer Learning non-trainable
- ResNet-like network

Precision, Recall and F1-score are weighted averages across four classes



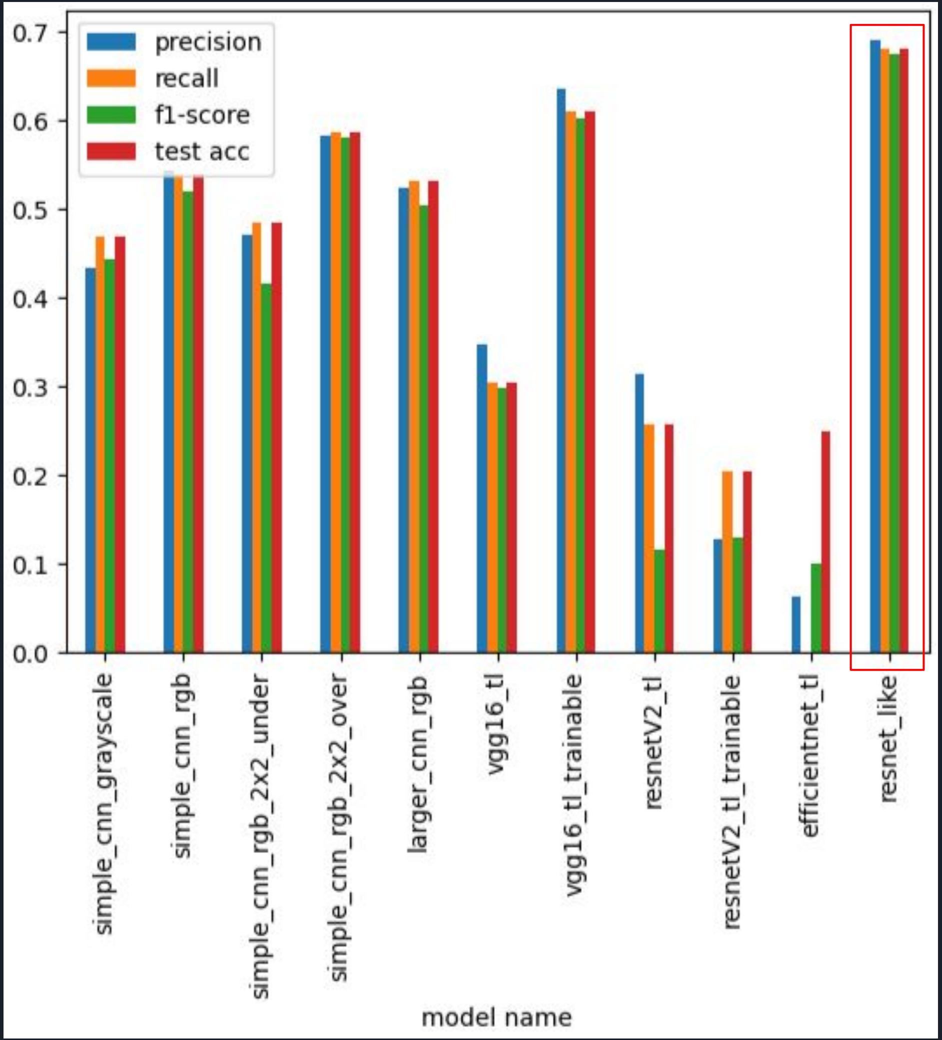
	model name	precision	recall	f1-score	test acc
0	simple_cnn_grayscale	0.433609	0.468750	0.443772	0.468750
1	simple_cnn_rgb	0.543023	0.539062	0.519088	0.539062
2	simple_cnn_rgb_2x2_under	0.470770	0.484375	0.415837	0.484375
3	simple_cnn_rgb_2x2_over	0.581464	0.585938	0.580455	0.585938
4	larger_cnn_rgb	0.523568	0.531250	0.503369	0.531250
5	vgg16_tl	0.347253	0.304688	0.298498	0.304688
6	vgg16_tl_trainable	0.634939	0.609375	0.601130	0.609375
7	resnetV2_tl	0.312992	0.257812	0.115780	0.257812
8	resnetV2_tl_trainable	0.127091	0.203125	0.129640	0.203125
9	efficientnet_tl	0.063492	0.000000	0.101266	0.250000
10	resnet_like	0.689200	0.679688	0.673484	0.679688

Model Development

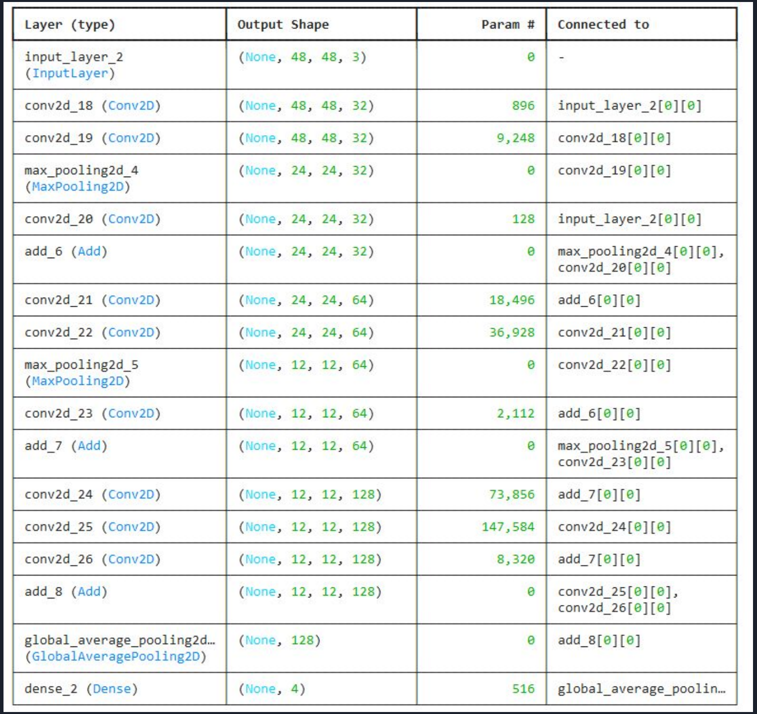
Comparison of Selected Models:

- Simple CNN Grayscale
- Simple CNN RGB
- Simple CNN RGB kernel 2x2 trained on undersampled data
- Simple CNN RGB kernel 2x2 trained on oversampled data
- Larger CNN RGB
- VGG16 Transfer Learning non-trainable
- VGG16 Transfer Learning trainable
- ResNetV2 Transfer Learning non-trainable
- ResNetV2 Transfer Learning trainable
- Efficient Net Transfer Learning non-trainable
- ResNet-like network

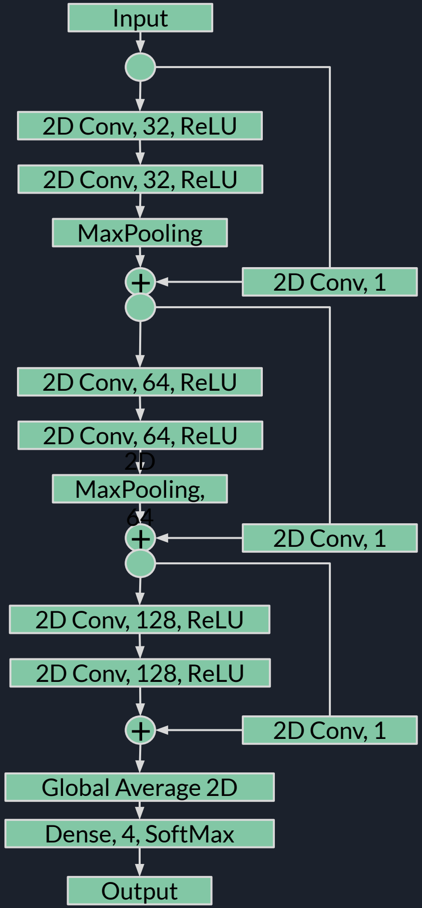
Precision, Recall and F1-score are weighted averages across four classes



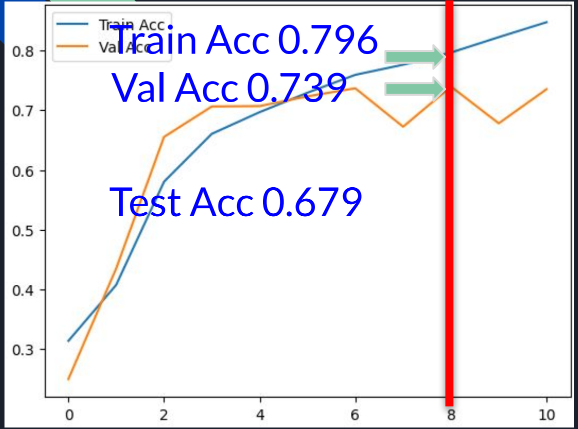
Model Development: Best Model Architecture



Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, 48, 48, 3)	0	-
conv2d_18 (Conv2D)	(None, 48, 48, 32)	896	input_layer_2[0][0]
conv2d_19 (Conv2D)	(None, 48, 48, 32)	9,248	conv2d_18[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 24, 24, 32)	0	conv2d_19[0][0]
conv2d_20 (Conv2D)	(None, 24, 24, 32)	128	input_layer_2[0][0]
add_6 (Add)	(None, 24, 24, 32)	0	max_pooling2d_4[0][0], conv2d_20[0][0]
conv2d_21 (Conv2D)	(None, 24, 24, 64)	18,496	add_6[0][0]
conv2d_22 (Conv2D)	(None, 24, 24, 64)	36,928	conv2d_21[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 12, 12, 64)	0	conv2d_22[0][0]
conv2d_23 (Conv2D)	(None, 12, 12, 64)	2,112	add_6[0][0]
add_7 (Add)	(None, 12, 12, 64)	0	max_pooling2d_5[0][0], conv2d_23[0][0]
conv2d_24 (Conv2D)	(None, 12, 12, 128)	73,856	add_7[0][0]
conv2d_25 (Conv2D)	(None, 12, 12, 128)	147,584	conv2d_24[0][0]
conv2d_26 (Conv2D)	(None, 12, 12, 128)	8,320	add_7[0][0]
add_8 (Add)	(None, 12, 12, 128)	0	conv2d_25[0][0], conv2d_26[0][0]
global_average_pooling2d… (GlobalAveragePooling2D)	(None, 128)	0	add_8[0][0]
dense_2 (Dense)	(None, 4)	516	global_average_poolin…



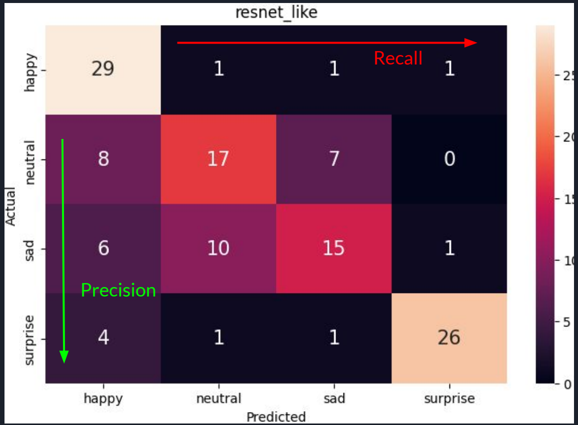
Model Development: Best Model Metrics

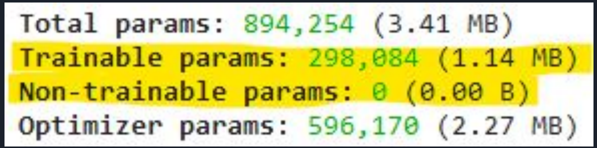


Best Model

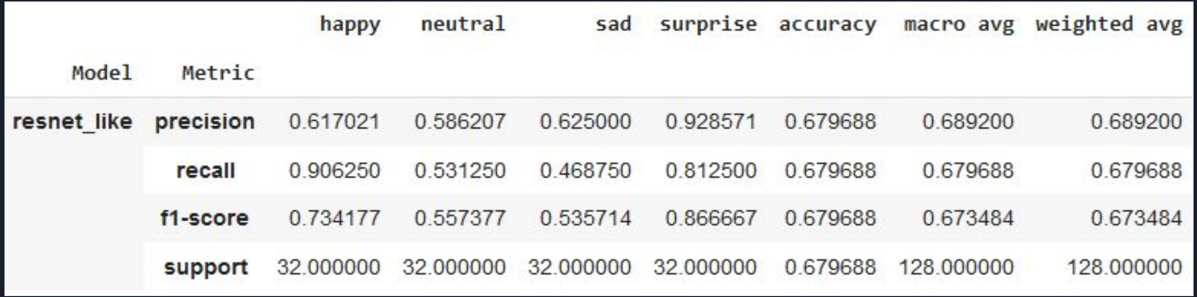
The best model, which was possible to find is ResNet-like model trained on RGB images.

Model size is 298,084 params, which can be consider small model. Size of the mode is only 1.14 Mb.





```
Total params: 894,254 (3.41 MB)
Trainable params: 298,084 (1.14 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 596,170 (2.27 MB)
```



		happy	neutral	sad	surprise	accuracy	macro avg	weighted avg
Model	**Metric**							
resnet_like	**precision**	0.617021	0.586207	0.625000	0.928571	0.679688	0.689200	0.689200
	recall	0.906250	0.531250	0.468750	0.812500	0.679688	0.679688	0.679688
	f1-score	0.734177	0.557377	0.535714	0.866667	0.679688	0.673484	0.673484
	support	32.000000	32.000000	32.000000	32.000000	0.679688	128.000000	128.000000

Model Development: Further Improvement

Way to improve model

- Collect more data
- Do in-depth analysis of image distribution across classes and create new dataset
- Use advanced augmentation techniques like AugMix
- Use more complex CNN architectures like Vision Transformers
- Use Transfer Learning from CNN trained on Emotion datasets
- Run more experiments with higher granularity for hyper parameters

Solution Recommendations

Solution Recommendations: ROI

The final goal of deploying Emotion recognition model is to develop successful application and gain profit.

From financial point of view this can be measured by Return on Investment (ROI) index. This index shows the ratio of net profit (Revenue - Cost) to the cost:

$$\text{ROI} = \frac{\text{Revenue - Cost}}{\text{Cost}}$$

Ideally, we would like to maximize the ROI, which can be done by minimizing Cost and maximizing Revenue. Here the consider way for cost reduction.

Solution Recommendations: Cost Optimization

Cost can be segregated into 2 big groups:

- Costs associated with Deep Learning model (Cost_DL)
- Other sources of expenses (Cost_other)

Cost = Cost_DL + Cost_other

Here we concentrate only Cost_DL.

Non-exhaustive list of sources of expense are as follows:

1. Model development cost: how much is cost to train a good model, including data collection and model development
2. Model deployment / exploitation:

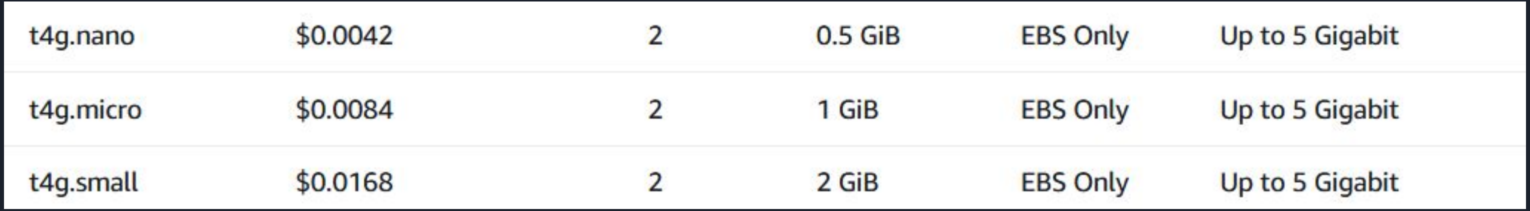
- Real-time prediction vs. on-demand (batch) prediction
- Is GPU is necessary for inference
- Response latency
- Deploy on-premise vs. deploy in a cloud
- Scalability

Solution Recommendations: Deployment/Exploitation

- Real-time vs. on-demand prediction: for real time prediction we need to keep end-point and pay all the time while end-point is alive. For on-demand, only pay for time to run predictions
- Is GPU is necessary for inference: if model is small then we can run model without GPU on a computer with low specs. If model is big , we need to use GPU to deliver inference in reasonable time.
- Response latency: if max latency is defined in Service Level Agreement (SLA), when it is necessary to meet this requirement. If model inference time does not make it possible, then we should either consider smaller model or use GPU-enabled resources to speed-up model inference.
- Deploy on-premise vs. deploy in a cloud: when deploying on premise, we have to consider Capital Expenses (CAPEX)[hardware, networking] + Operational Expenses (OPEX)[electricity and water bills, etc.]. In the case of cloud we only pay for used time, without owing hardware and networking infa. Thus, we only face OPEX.
- Scalability: size of the model is determines size of Docker container and thus max number of container running together given the limited cluster resources

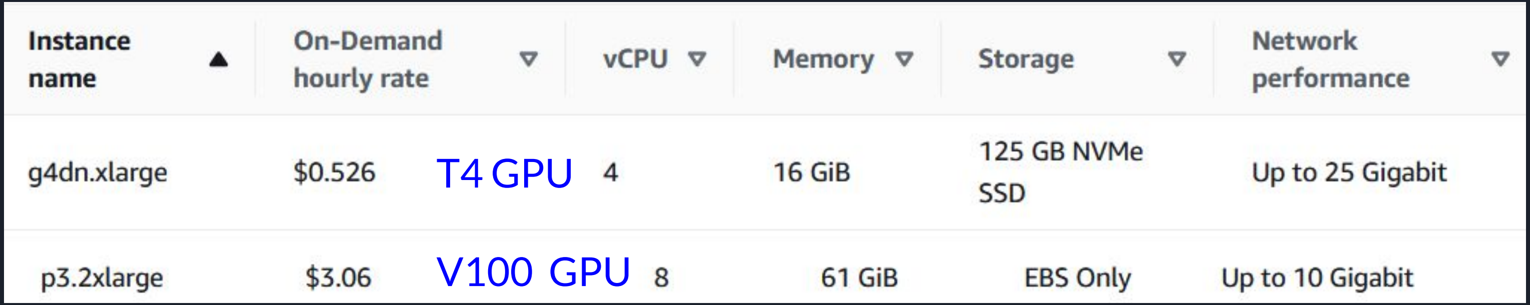
Solution Recommendations: Prices for GPU vs. Non-GPU processing

- To illustrate difference cost of GPU vs. Non-GPU resource, we will use example from AWS cloud. Below, there is a cost per hour of different instances:
- Non-GPU



t4g.nano	$0.0042	2	0.5 GiB	EBS Only	Up to 5 Gigabit
t4g.micro	$0.0084	2	1 GiB	EBS Only	Up to 5 Gigabit
t4g.small	$0.0168	2	2 GiB	EBS Only	Up to 5 Gigabit

- GPU



Instance name ▲	On-Demand hourly rate ▽		vCPU ▽	Memory ▽	Storage ▽	Network performance ▽
g4dn.xlarge	$0.526	T4 GPU	4	16 GiB	125 GB NVMe SSD	Up to 25 Gigabit
p3.2xlarge	$3.06	V100 GPU	8	61 GiB	EBS Only	Up to 10 Gigabit