COL 341 : Salt Identification

-- Pratyush Maini (2016CS10412)
--Shashank Goel (2016CS10332)

Salt Identification:

We were supposed to solve a Salt Identification Challengea. To create the most accurate seismic images and 3D renderings, we aim to build an algorithm that automatically and accurately identifies if a subsurface target is salt or not.

Architectures Tried:

1. U-Net :

The **U-Net** is a convolutional neural network that was developed for **biomedical image segmentation** and has received much appreciation in the ML community. Since the given problem requires a distinction between images having salt or not, which relies on segment identification in seismic images, using a U-Net was a natural choice.
The network consists of a **contracting path and an expansive path,** which gives it the **u-shaped architecture**. The contracting path consists of repeated application of **convolutions**, each followed by a **ReLU** layer and a **max pooling** operation.
During the contraction, the **spatial information is reduced** while **feature information is increased**. The expansive pathway combines the feature and spatial information through a sequence of convolutions & concatenations with high-resolution features from the contracting path

2. Resnet-50:

We observed a **saturation (and some degradation) in accuracy** with increased layers. So, we decided to use a **Residual Learning framework** to preserve good results through a network with many layers. The deep residual network deals with these problems by using residual blocks, which take advantage of residual mapping to preserve inputs.
The neural network collapses into fewer layers in the initial phase, which makes it easier to learn, and thus **gradually expands the layers** as it learns more of the

feature space. During later learning, when all layers are expanded, it will stay closer to the manifold and thus learn faster.
On the other hand, a neural network without residual parts will explore more of the feature space. This makes it more **vulnerable to small perturbations** that cause it to leave the manifold altogether, and **require extra training data to get back on track**.

Other Techniques used and their Explanation:

A- **Use of Coverage :** New feature that tells the percentage of the whole image that has salt in it.
B - **Ad Hoc Classification**: Assign classes based on percentage of salt.

1. **Cosine Annealing: Stochastic Gradient Descent** with Warm Restarts is common in gradient-free optimization to deal with multimodal functions. Partial warm restarts are also gaining popularity in gradient-based optimization to improve the rate of convergence in accelerated gradient schemes to deal with ill-conditioned functions. In this paper, we propose a simple warm restart technique for stochastic gradient descent to improve its anytime performance when training deep neural networks.

2. **Training Set Flip & Data Augmentation :** We used Numpy's Fliplr function to flip the training images and then augment it back to the training data. This technique helps in increasing the diversity in the dataset and makes the trained model more robust to new test data. We also experimented with image rotation by 10 deg on either side to obtain newer samples.

3. **Early Stopping:** We varied patience values over a range of experiments to find the optimum stopping conditions, while at the same time not overfitting on the training set.

4. **Plateau LR Reduction:** Learning rate would be reduced depending on the maximum and minimum learning rates on plateau detection. This detection sensitivity was varied for different patience values.

5. **Batch Size Variations:** Varying batch size was seen to create a difference of +- 1 percent on accuracy score. We varied batch size in {32, 64, 128} for our experiments.

6. **Analysis of different Loss Functions:**

A. ***Binary Cross-entropy:***
The binary cross-entropy loss function has an inbuilt implementation in **Keras** which we used in our code. We used it as an alternative to squared error. Cross-entropy can be used as an error measure when a network's outputs can be thought of as representing **independent hypotheses**, and the **node activations** can be understood as representing the **probability** that **each hypothesis** might be true. It is useful in problems in which the targets are 0 and 1.

B. ***Focal Loss:***
Its implementation was based on "Focal Loss for Dense Object Detection" (Lin et. al.). We were facing extreme foreground-background class imbalance during training of dense detectors which was causing inaccuracies in test results. We addressed this class imbalance by **reshaping the standard cross entropy loss such that it down-weights the loss assigned to well-classified** examples. **Focal Loss** focuses training on a sparse set of hard examples and prevents the vast number of easy negatives from overwhelming the detector during training.

C. ***Lovasz Loss:***
The Jaccard index, also referred to as the **intersection-over-union score**, is commonly employed in the evaluation of image segmentation results given its perceptual qualities, scale invariance - which lends appropriate relevance to small objects, and appropriate counting of false negatives, in comparison to per-pixel losses. Lovasz Loss helps optimize this metric by its direct evaluation using the Lovasz Hinge approximation. The loss is shown to perform better with respect to the Jaccard index measure than the traditionally used cross-entropy loss.

<u>Final Model:</u>

The training of our model consists of the following three phases:
Training set size = 7200
Validation set size = 400

Test set size = 400

Phase 1: Base Training

- Initial build: Unet + resnet34
- Loss: binary_crossentropy
- Optimizer: Adam
- Learning Rate: 0.01
- Epochs: 105
- Batch size: 32
- Early Stop: patience = 20
- Reduce LR on plateau: factor: 0.5, patience = 5, Min_LR = 0.0001

Phase 2: Fine Tuning 1

- pretrained weights: best checkpoint of Phase 1
- Loss: lovasz loss
- Optimizer: Adam
- Learning rate: 0.01
- Epochs: 50
- Batch size: 32
- Early Stop: patience = 30
- Reduce LR on plateau: factor: 0.5, patience = 6, Min_LR = 0.00001

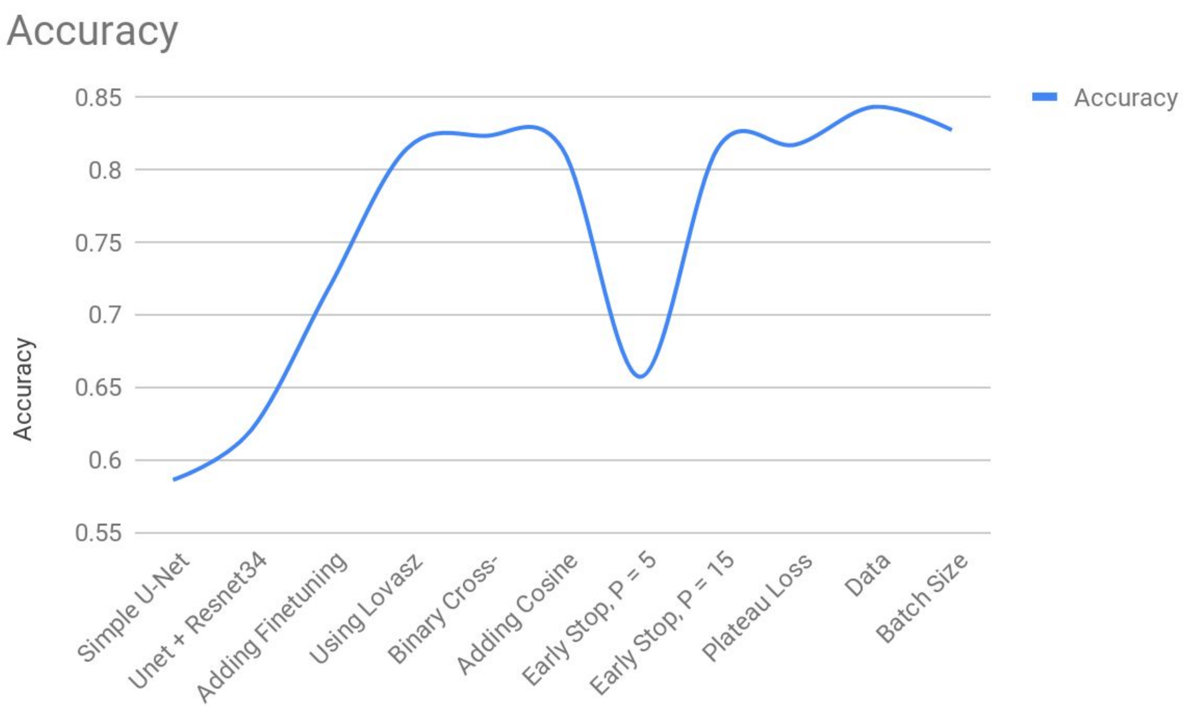
Phase 3: Fine Tuning 2

- pretrained weights: best checkpoint of Phase 1
- Loss: lovasz loss
- Optimizer: SGD
- Learning rate: 0.005
- Momentum: 0.9
- Epochs: 30
- Batch size: 64
- Cosine Annealing Scheduler
- Reduce LR on plateau: Min_LR = 0.0001, Max_LR = 0.01

Accuracy Results:

	Accuracy
Simple U-Net	0.586
Unet + Resnet34	0.62
Adding Finetuning	0.718

Using Lovasz Loss	0.814
Binary Cross-Entropy for Baseline	0.823
Adding Cosine Annealing	0.814
Early Stop, P = 5	0.657
Early Stop, P = 15	0.815
Plateau Loss	0.817
Data Augmentation	0.843
Batch Size Variation	0.827



Note: We chose the given architecture since we had to find the best balance between an architecture that gave maximum score, and one that trained on HPC using 1 GPU & 4 CPUs in 8 hours (which was one of the pre-specified conditions).