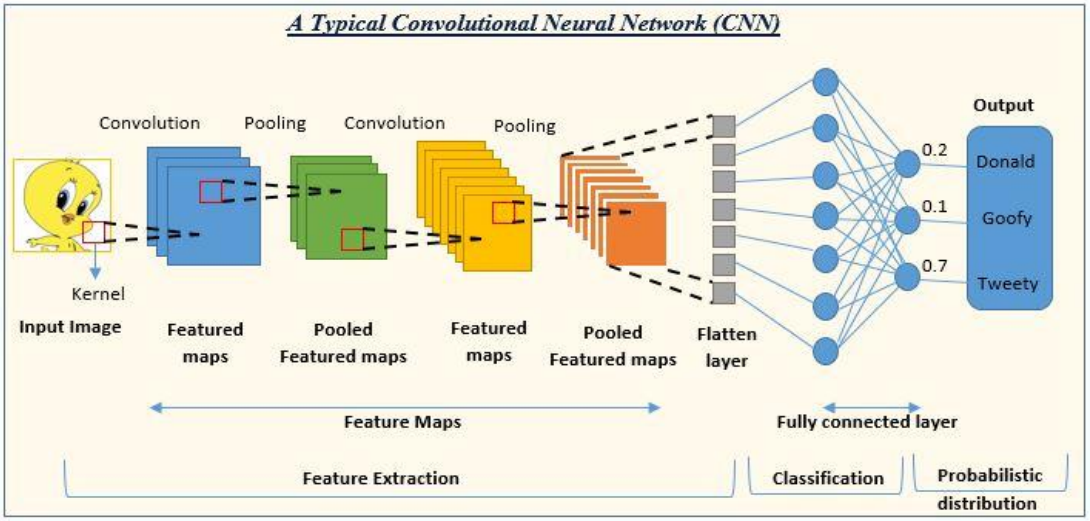
VR MINI PROJECT

Assignment 3a: Play with CNNs

Ans:

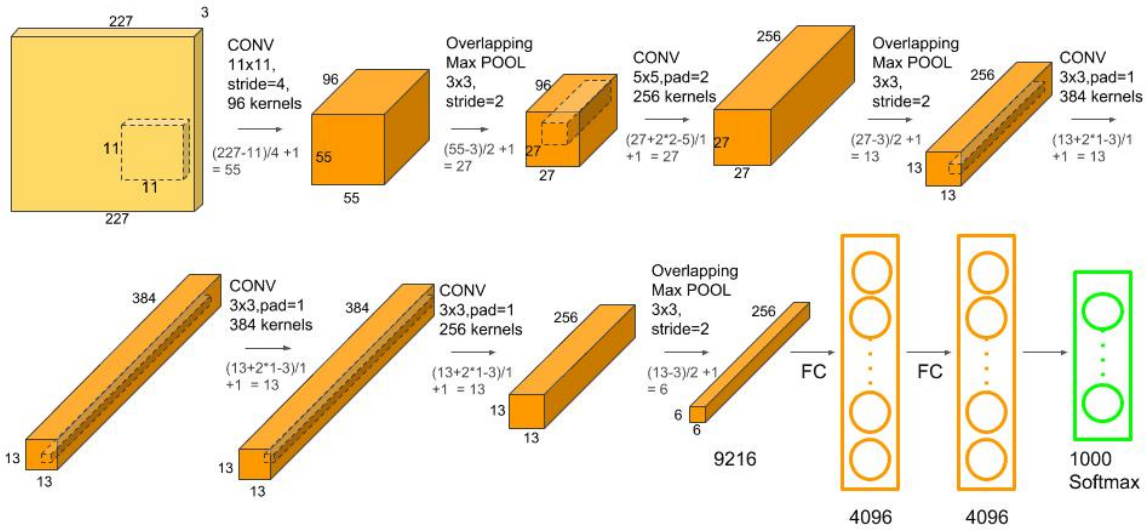
What is CNN: CNN stands for Convolutional Neural Network, which is a type of neural network commonly used in computer vision tasks such as image and video recognition, object detection, and segmentation.

The key difference between a CNN and a traditional neural network is the way in which it processes data. A CNN applies a series of convolutional filters to the input data, which helps to extract features and patterns from the input image.These filters typically start by detecting simple features such as edges, lines, and corners, and then gradually progress to more complex features such as textures and shapes.After the convolutional layers, a CNN usually includes pooling layers, which downsample the feature maps produced by the convolutional layers and help to reduce the dimensionality of the data and get the global perspective of data.Finally, the output from the convolutional and pooling layers is fed into one or more fully connected layers, which perform classification or regression tasks based on the extracted features.



AlexNet: AlexNet is a deep convolutional neural network (CNN) architecture designed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012. It won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 by achieving a top-5 error rate of 15.3%, which was significantly better than the second-best performance of 26.2%.

The architecture of AlexNet consists of 5 convolutional layers, followed by 3 fully connected layers and a softmax output layer. The network has a total of 60 million parameters and 650,000 neurons. The convolutional layers use a rectified linear unit (ReLU) activation function, and the network uses dropout regularization to prevent overfitting.



Architecture Of Alexnet

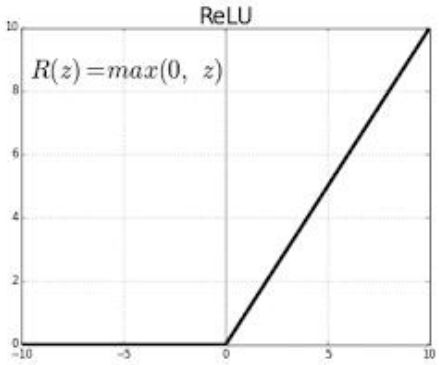
AlexNet was significant in advancing the field of computer vision because it demonstrated the power of deep learning for image recognition tasks and paved the way for subsequent breakthroughs in the field.

Training A CNN: While training a RCNN there are few things we have to keep in our minds like: what is the best architecture of CNN for our task , which loss function to use according to our task , which activation function to use in our architecture , which optimizer to use in our gradient descent optimization algorithm for faster convergence and less training time.

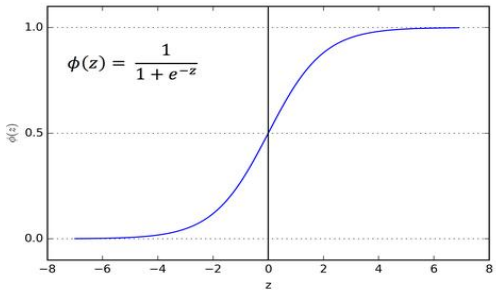
For classification task we generally use cross-entropy loss and for regression task we generally use mean squared error loss.

The 3 most popular activation functions are ReLU, Sigmoid & tanh.

ReLU: The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.



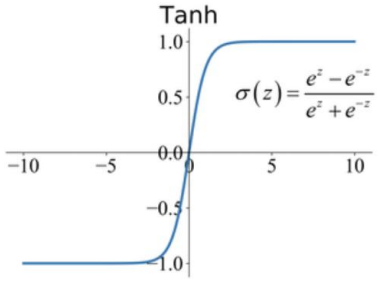
Sigmoid :The Sigmoid Function curve looks like a S-shape.The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output.



Sigmoid

Tanh:

Tanh is also like logistic sigmoid but better. The range of the Tanh function is from (-1 to 1). Tanh is also sigmoidal (s - shaped).The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the Tanh graph.



All the functions are differentiable and monotonic.

There are several optimizers available for deep learning, each with its own advantages and disadvantages. Some of the commonly used optimizers are:

Stochastic Gradient Descent (SGD): SGD is the most widely used optimizer in deep learning. It updates the parameters in the direction of the negative gradient of the loss function for each mini-batch of data.

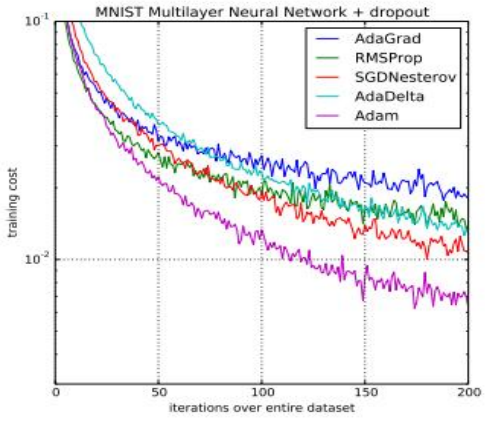
Adam: Adam is a popular optimizer that uses a combination of adaptive learning rates and momentum. It adapts the learning rate for each parameter based on the magnitude of its gradient and past gradients.

Adagrad: Adagrad adapts the learning rate of each parameter based on the sum of the squares of its past gradients. It is well-suited for sparse data.

RMSProp: RMSProp is similar to Adagrad but it uses a moving average of the squared gradient instead of the sum of the squares. It is also well-suited for sparse data.

Adadelta: Adadelta is similar to RMSProp but it uses a moving average of the squared gradient and the squared parameter update.

Each optimizer has its own hyperparameters that need to be tuned to obtain the best performance on a given dataset.Today Adam is the most popular optimizer in deep learning.



Implementation Details:

Dataset Details: We used CIFAR-10 dataset for our project.The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.The 10 classes are airplane, car, bird, cat, deer, dog, frog, horse, ship, and truck.

Preprocessing: The following preprocessings were done on CIFAR-10 dataset:
Random Crop: The first preprocessing step is to randomly crop the input image to a size of 32x32 with padding of 4 pixels on each side. This is done to augment the data and create more diverse images for training.
Random Horizontal Flip: The second preprocessing step is to randomly flip the image horizontally with a probability of 0.5. This is another form of data augmentation to create more diverse images and help the model generalize better to unseen data.
ToTensor: The third preprocessing step is to convert the image to a PyTorch tensor. PyTorch uses tensors as the main data structure for its deep learning models, and this step converts the image from its original format to a tensor that can be processed by the model.
Normalize: The final preprocessing step is to normalize the tensor values using mean and standard deviation values specific to the CIFAR-10 dataset. This helps to reduce the range of the input data and make it easier for the model to learn the patterns in the data.
While training we divided our dataset into batches of size 64 and we used cross-entropy loss for our classification task.

Used Model Architectures:

We tried several model architectures for our task:

1.**PandaNet1**: It takes a 32x32 pixel input image with 3 color channels (RGB) and outputs a probability distribution over 10 classes.
The architecture consists of 3 convolutional layers followed by max-pooling layers to reduce the spatial dimensions of the output. Each convolutional layer uses a 3x3 kernel with padding of 1 to maintain the spatial resolution. The first layer has 32 output channels, the second has 64, and the third has 128.
After the final pooling layer, the output is flattened and passed through two fully connected layers (or linear layers) with a ReLU activation function applied to the first layer and a dropout regularization with a probability of 0.5. The first fully connected layer has 512 hidden units and the final layer has 10 output units, corresponding to the 10 classes of the classification problem. Finally, a softmax activation function is applied to the output layer to obtain the class probabilities.
The network has a total of **1,147,466** trainable parameters, which include the weights and biases of the convolutional layers, the weights and biases of the fully connected layers, and the probabilities for each class in the output layer.

2. PandaNet2: This CNN takes a 3-channel image as input and consists of 3 convolutional layers followed by max-pooling layers, and 3 fully connected layers. ReLU activation function is used after each layer except the last fully connected layer which uses log-softmax activation function. The first convolutional layer has 64 filters, the second layer has 128 filters, and the third layer has 256 filters. Each convolutional layer uses a 3x3 kernel with stride 1 and padding 1 to preserve the spatial dimensions of the input. The max-pooling layers have a 2x2 kernel with stride 2 to downsample the spatial dimensions by a factor of 2. After the convolutional and max-pooling layers, the output is flattened and passed through two fully connected layers with 256 and 128 units respectively, each followed by a ReLU activation function. The final layer is a fully connected layer with 10 units (corresponding to the 10 output classes) and uses log-softmax activation function to normalize the output probabilities.This model has **930826** total trainable parameters.

3. PandaNet3: It consists of six convolutional layers followed by max-pooling layers, three fully connected (linear) layers, and ReLU activation functions between layers. Batch normalization is used to improve the convergence speed and regularization. The number of feature maps (depth) increases from 32 to 256 gradually, while the spatial resolution is reduced by the max-pooling layers. The output of the last linear layer is passed through a log-softmax activation function, which is commonly used for multi-class classification. This architecture is not a specific famous model, but it follows the general design principles of popular CNNs like VGG, ResNet, and Inception. It has total **5,852,234** trainable parameters.

4. PandaNet with Skip Connections: It consists of several layers of convolutional and residual blocks, followed by a classifier. The architecture takes 3-channel input images and outputs a class prediction among 10 possible classes.The CNN architecture consists of the following layers:
The first layer is a convolutional block with 64 output channels.
The second layer is another convolutional block with 128 output channels and a max-pooling layer with a kernel size of 2.
The third layer is a residual block consisting of two convolutional blocks with 128 output channels.
The fourth layer is another convolutional block with 256 output channels and a max-pooling layer with a kernel size of 2.
The fifth layer is another convolutional block with 512 output channels and a max-pooling layer with a kernel size of 2.
The sixth layer is a residual block consisting of two convolutional blocks with 512 output channels.
The seventh and final layer is the classifier which consists of a max-pooling layer with a kernel size of 4, a flattening layer, and a linear layer that outputs the class scores.
The model uses the ReLU activation function after each convolutional block, and it also uses batch normalization to improve training stability. It has total **6,575,370** trainable parameters.

Training Details & Results:

Sr. NO.	Network	Activation	Optimizer	No. Epochs	Training Time (s)	Val. Accuracy	Test Accuracy
1	PandaNet-1	ReLU	SGD	50	912.53	24.86	25.55
2	PandaNet-1	Sigmoid	SGD	50	401.80	10.28	11.56
3	PandaNet-1	Tanh	SGD	50	404.41	25.3	27.63
4	PandaNet-1	ReLU	Adam	20	372.99	62.74	63.12
5	PandaNet-1	Tanh	Adam	20	384.69	66.88	67.59
6	PandaNet-1	Sigmoid	Adam	20	564.71	53.32	54.02
7	PandaNet-1	ReLU (He initialized)	Adam	20	567.67	10.10	10.0
8	PandaNet with Skip Connections	ReLU	SGD with momentum	40	1320.63	84.04	83.64
9	PandaNet-2	ReLU	SGD with momentum	40	1118.97	78.36	78.34
10	PandaNet-2	Tanh	SGD with momentum	40	326.46	76.88	77.51
11	PandaNet-2	Sigmoid	SGD with momentum	40	325.72	10.20	10.00

12	PandaNet-2	ReLU	Adam	40	325.29	82.12	82.6
13	PandaNet-2	Tanh	AdaDelta	40	324.02	65.04	65.51
14	PandaNet-3	ReLU	SGD with momentum	50	1532.57	77.08	77.95
15	PandaNet-3	ReLU	NAG	50	1112.52	80.74	80.46
16	PandaNet-3	ReLU	AdaDelta	50	1120.35	78.36	78.57
17	PandaNet-3	ReLU	AdaGrad	50	1102.49	77.94	77.27
18	PandaNet-3	ReLU	RMS_Prop	50	1099.29	84.04	84.67
19	PandaNet-3	ReLU	Adam	50	1091.60	85.32	85.81
20	PandaNet-3	Tanh	Adam	50	423.98	69.94	69.69
21	PandaNet-3	Sigmoid	RMS_Prop	50	409.40	85.02	84.96

Obseravations:

- We observe that among all the activation functions ReLU performs the best with all the models.
- We also notice that among all the activation functions, Sigmoid performs the worst in most of the cases.
- Adam optimizer performs well as it can converge the model in less number of epochs in less amount of time.
- We also noticed that adding skip-connection to our model, improves the model's performance.
- PandaNet2 (with ReLU and Adam) performed really well even though having the least number of parameters among the 3 models that we have tested. It's best performance was with an accuracy of 82.6 on the test data in 40 epoch as compared to our best model PandaNet3 in 50 epochs with 85.81 accuracy.
- PandaNet1's best performance was 67.59 accuracy with Tanh activation and Adam optimizer.

Recommended architecture:

Based on our exploration on all the types of activation functions and optimizers and model architectures, **PandaNet-3** is our recommended architecture with **Adam** optimizer and **ReLU** activation.