DL Assignment 2: Convolution Neural Networks

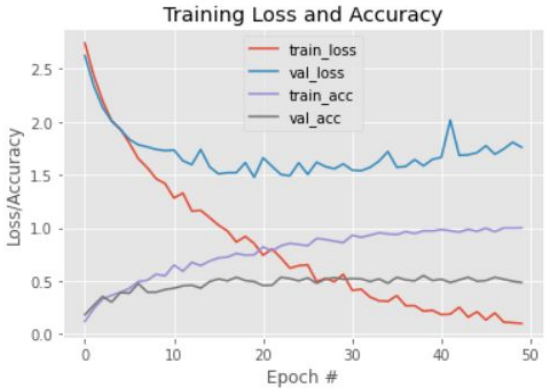
Submitted By: Phalguni Rathod | Student No. : R00183770

Part A: CNN from scratch & variety

Part A - i: Baseline model, extension of baselines and Data Augmentation

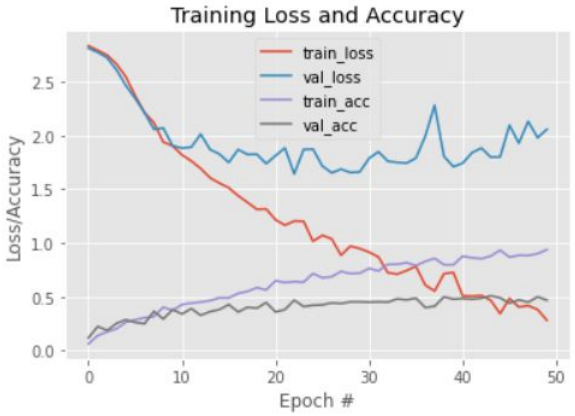
Baseline & Extensions

1. Base Line:



We can observe that the model is trained for 50 epochs, and it starts to overfit very early during the training, which is around 7-8th Epoch and further diverges exponentially within a few more epochs.
Its effect can be seen on the train-validation accuracy where training reaches accuracy reaches near 100% while validation is only 50% for 50 epochs.

2. Baseline Extension 1:



Here, we have extended the baseline by adding 1 more layer to it: Convolutional & MaxPooling layer. We kept the lr and epochs the same as earlier, i.e 0.01 & 50 respectively.
Overfitting: Starts post 12th epoch and train-validation plots diverges quickly.
The effect of the overfitting can be observed in train-val accuracy where train curve increases consistently, while validation gets staged where it is(around .5)

3. Baseline Extension 2:

Here, we have extended the baseline by adding 2 more layers to it: Convolutional & MaxPooling layer * 2. We kept the lr and epochs the same as earlier, i.e 0.01 & 50 respectively.
Overfitting: Starts post 20th epoch and train-validation plots diverges quickly. The effect of the overfitting can be observed in train-val accuracy where train curve increases consistently, while validation gets stuck where it is(around .5)

4. Baseline Extension 3:

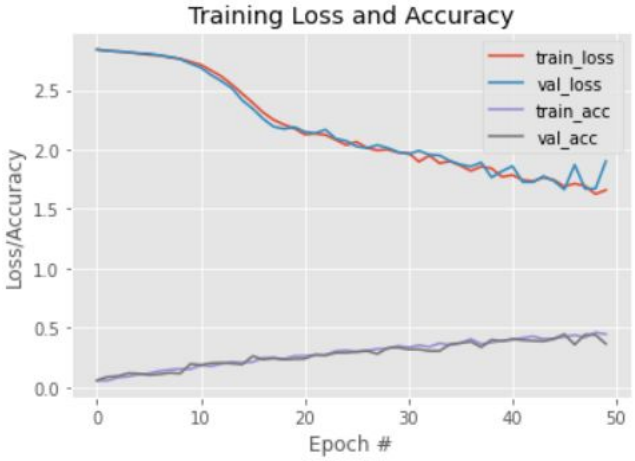
Here, we have extended the baseline by adding 3 more layers to it: Convolutional & MaxPooling layer * 3. We kept the lr and epochs the same as earlier, i.e 0.01 & 50 respectively.
Overfitting: Starts post 30th epoch and train-validation plots diverges quickly. The effect of the overfitting can be observed in train-val accuracy where train curve increases started diverging from the validation curve around the 50th epoch.

Observation,Comparison & Contrast of above models:

1. The structure/flow of curves in all the 4 models is similar.
2. With increase in the depth of network, the overfitting of the model is delayed to later epochs, i.e Baseline overfits at 8th epoch, BL Ext 1 overfits at 12th epoch, BL Ext 2 overfits at 20th Epoch, BL Ext 3 overfits at 30th epoch.
3. The divergence of train-validation accuracy is abrupt in baseline

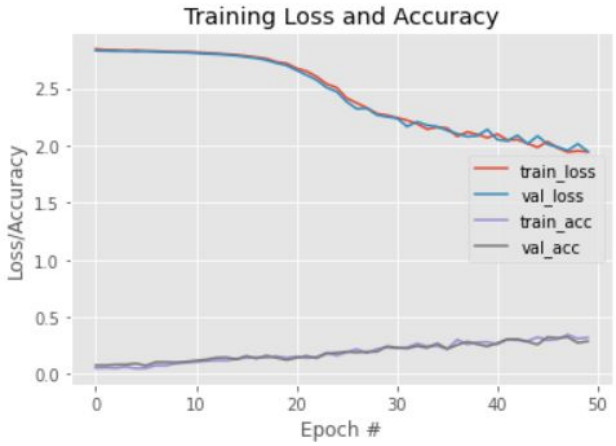
Data Augmentation on 2 deepest model of above created baselines

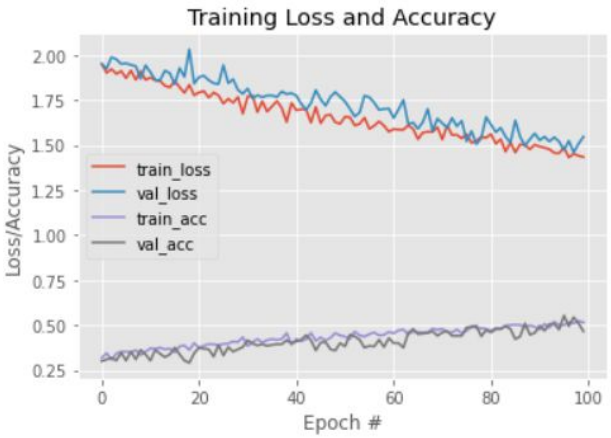
1. Augmentation on Baseline extension 2:



The data augmentation is executed twice, first with 50 epochs and we see that the graph is pretty stable, to explore more about its further behaviour we executed it for 100 epochs.
The graph doesn't overfit in plot 1 at all, observe the start of overfitting see the divergence in train-validation curve of post 70th epoch. The second is very unstable. It can be observed the train-val accuracies are around .50 for both the plots.

2. Augmentation on Baseline extension 3:





The data augmentation is executed twice, first with 50 epochs and we see that the graph is pretty stable, to explore more about its further behaviour we executed it for 100 epochs.
The graph doesn't overfit in plot 1 at all, neither does it start overfitting in plot 2 but a lot of instability. It can be observed the train-val accuracies are around .30-.40 first plot and increases slowly to 0.6 for 100 epochs in plot 2.

Observation, Comparison & Contrast of above models:

1. Using augmented data for training and validation had a very strong and positive impact on the models
2. We can observe the BL extension 2 and Aug of Ext 2 have very distinct and measurable differences,
    a. Earlier it was overfitting very early around 20th epoch while post augmentation it started diverging around 80th epoch which is just a start and not abrupt divergence in earlier case
    b. The accuracy which was earlier started diverging for train and validation, post augmentation is at par with each other no divergence is observed.
3. As above, we can observe the BL extension 3 and Aug of Ext 3 also have very distinct and measurable differences,
    a. Earlier it was overfitting very early around 30th epoch while post augmentation it didn't overfit at all at least till 100th epoch.
    b. The accuracy which was earlier started diverging for train and validation, post augmentation is at par with each other no divergence is observed until 100th epoch.
4. This could be because, by adding augmented data we are giving the model an opportunity to learn better and more about the images that are being fed to it.
5. Augmentation gives the model the ability to generalize. It makes the model more robust.

Part A - ii Ensembling: Fixed Arch & Pre-trained Arch

Fixed Arch

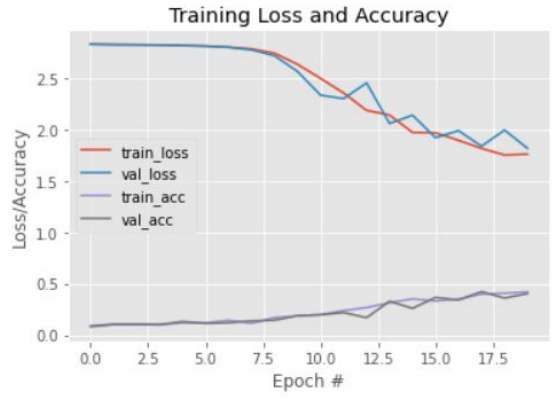
We are using Shallow VGG Arch, similar to what was taught during the lecture. We have

```
visible = Input(shape=(128, 128, 3))
conv1 = Conv2D(32, kernel_size=4, activation='relu')(visible)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
conv2 = Conv2D(16, kernel_size=4, activation='relu')(pool1)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
flat = Flatten()(pool2)
hidden1 = Dense(10, activation='relu')(flat)
output = Dense(1, activation='sigmoid')(hidden1)
model = Model(inputs=visible, outputs=output)
```

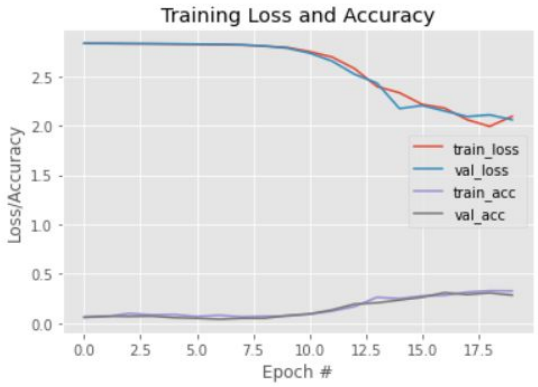
made 5 base learners with arch. Yet all 5 base learners learn the data differently as every learner has their own different set of weights.

When the output of all the learners is aggregated, by taking mean of it, it means ensembling them. Epochs = 20, Learning Rate: 0.01, Optim: SGD

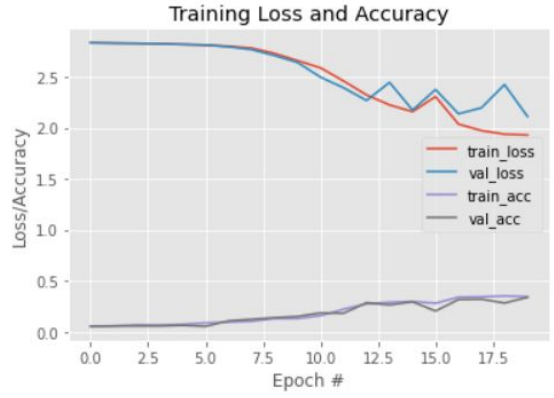
Model # 1



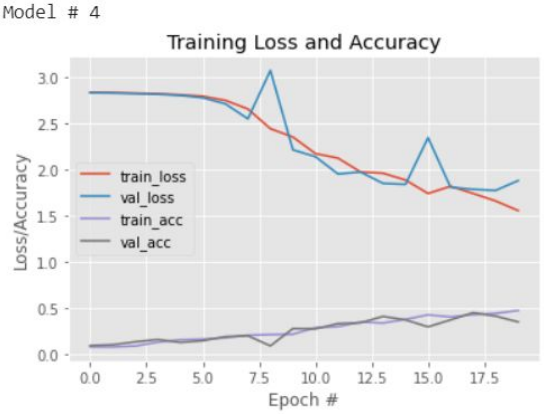
Model # 3



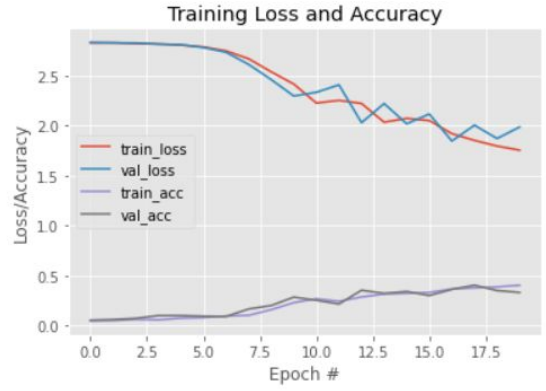
Model # 2



Model # 4



Model # 5




```
Ensemble Model and accuracy 0.3911764705882353
Model # 1 and accuracy 0.40294117647058825
Model # 2 and accuracy 0.3411764705882353
Model # 3 and accuracy 0.2911764705882353
Model # 4 and accuracy 0.34705882352941175
Model # 5 and accuracy 0.32941176470588235
```

Observation:

- As we have consistently modelled each time with 20 epochs, we have kept it the same here, but it looks the model didn't train well.
- In all the base learners, the train-val loss has been consistent, while train-val accuracy is consistently broadly, but have few spikes in Model #2, Model #4
- On Ensembling all the base learner and testing of test set, we don't get desired accuracy with any of the base learner individually or ensembled
- This could be due the less #Epochs used for training, yet so, low accuracy was not expected.

To overcome this, we also ensembled using, Preitrained models.

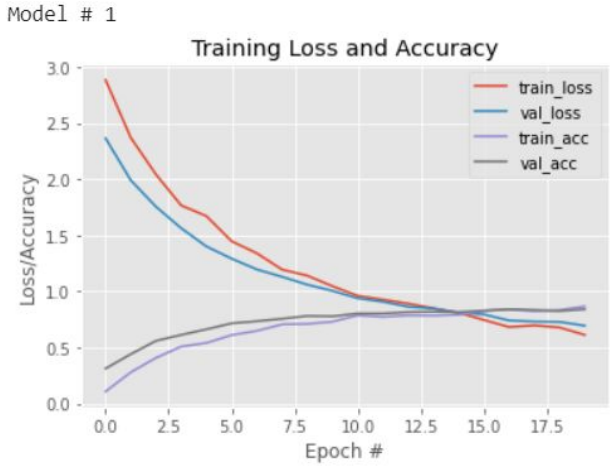
Pre-Trained Model Arch:

Models Used:

1. ResNet50V2
2. DenseNet121
3. MobileNetV2

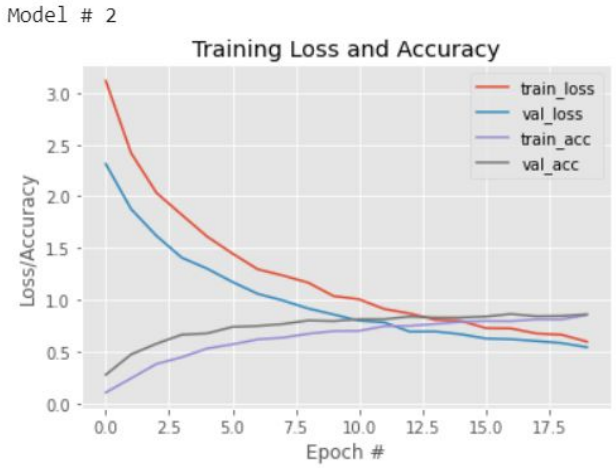
We are also creating checkpoints to save the best model with minimal weights, this has been done in the colab, folder structure with best weights is zipped and shared with submission files.

Model 1:



Observation:

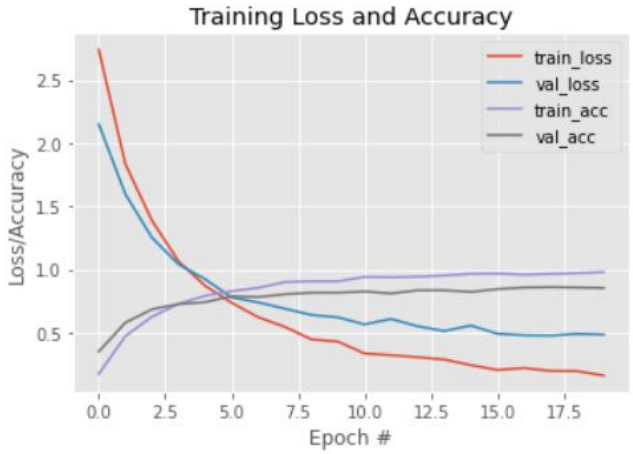
- This is ResNet50V2, and performs really well in 20 epochs.
- It reaches the accuracy of .80s with 20 epochs and trend show it to be doing well with increase in epochs later.
- No sign of overfitting is seen



Observations:

- This is DenseNet - Performs Well
- This performs well but slower than ResNet, nonetheless it is not bad at all.
- Seems to be converging somewhere in 20s, or early thirties.
- Don't see overfitting soon.

Model # 3



Observation:

- This is MobileNet and performs good but diverges quickly
- Starts overfitting the data somehere around 10th epoch itself
- While other other model train better than this, making this model less suitable of the dataset.

Ensemble:

```
Ensemble Model and accuracy 0.9323529411764706
Model # 1 and accuracy 0.8411764705882353
Model # 2 and accuracy 0.8558823529411764
Model # 3 and accuracy 0.8558823529411764
```

- When testing on the ensemble by taking mean of the output of each model, we get really, really amazing model.
- Ensemble outperforms all the base models by 8-9%
- This ensemble outperforms the fixed arch ensemble with same parameters but different arch by, 40-45%

Ensembling is basically a technique where we use the power of individuals and make the super power out of it, just like Avengers!

Part B: Transfer Learning

It is a concept where we leverage the knowledge of the models which are already trained on some other datasets belonging to the same area as the current one like pre-trained models on images will be used for image related current model or nlp related will be used accordingly.
Here, we basically transfer the learning a model had by training on some other dataset, by implementing it on our current dataset.
There are 2 major ways of this: Feature Extraction (subpart - i) and Fine tuning (subpart - ii)

Part B - i : Feature Extraction

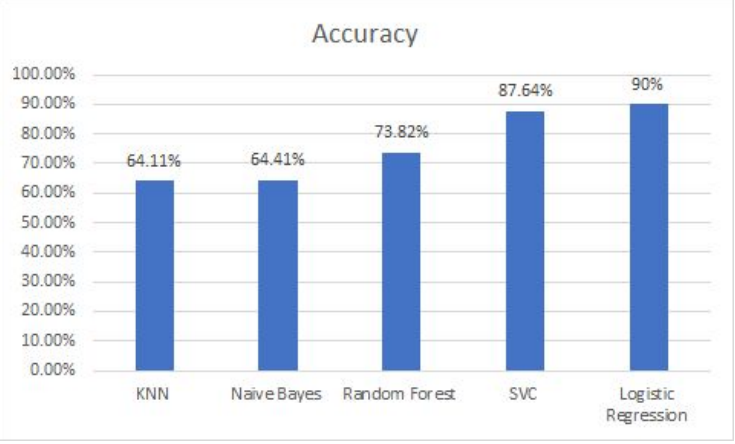
1. Base Model/Feature Extractor: ResNet50

   ResNet50 has been used as a feature extractor or base model. The intention behind is that, resnet have the skip connections, which means features can be propagated for further layers.
   Batch Normalization is used as the core of Resnet. Input Layers are adjusted accordingly to increase the performance of the network.
   Use of the Identity Connection, protect the network from vanishing gradient problems.[https://tech.zegami.com/comparing-pre-trained-deep-learning-models-for-feature-extraction-c617da54641]

   Weights of base_model have been set to use "imagenet", as this will really help in extracting more variety of features. Because ImageNet contains variety of objects.

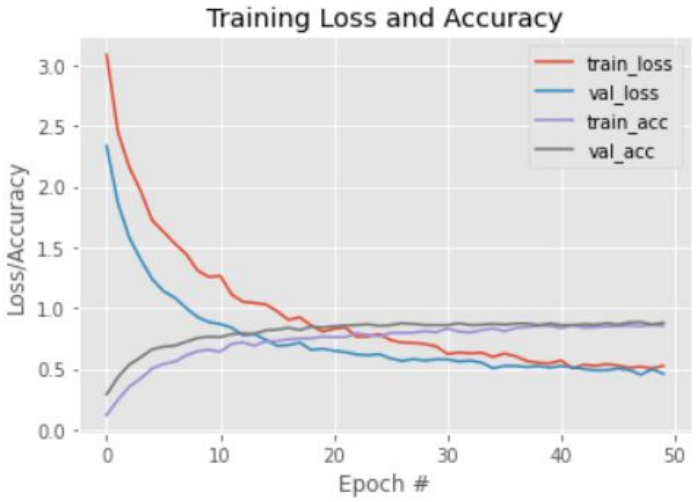
   We have kept include_top as False, as we don't want the FC & Softmax layer at the end of the model, we just want the features which we will further use to train the ML Algo.

2. Secondary/ML Algo:
   a. Random Forest - Acc: 73.82%
   b. Logistic Regression - Acc: 90%
   c. Naive Bayes - Acc: 64.41%
   d. KNN - Acc: 64.11%
   e. SVC - 87.64%



Part B - ii : Fine Tuning

1. **Phase A:**
   In this phase we cut off the FC and softmax layers on the pre-trained model and add new FC Layer and softmax having neurons same as # of classes to be predicted.
   We are loading augmented data for training, we are using SGD optimise with learning rate as 0.01 and running it for 50 epochs.

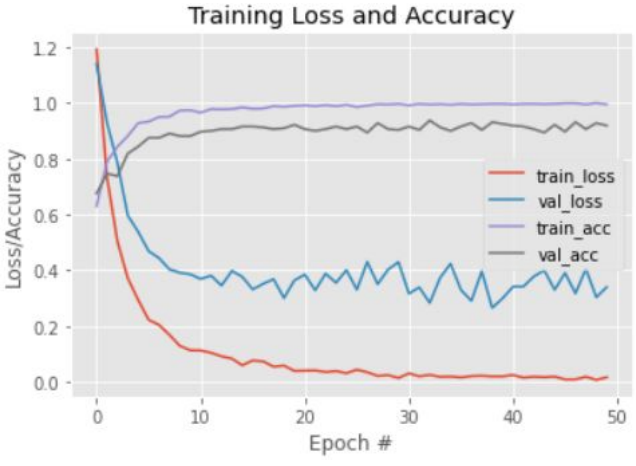


Observations:

1. It's really a nice curve for train - validation loss as well as train - validation accuracy.
2. We can easily observe the convergence of both curve pot 25th epoch.
3. I expect it perform really well and reach good accuracy if increase the # of epochs.
4. We can't comment on overfitting given the current plot for 50 epochs, as the curves have just started converging their divergence is something we can't comment on.

2. **Phase B:**

Fine Tuning with unfreezing layers from base model. We have unfreezed all the layers of ResNet50V2 and now are training with extended layers of FC and softmax. We are loading augmented data for training, we are using SGD optimise with learning rate as 0.01 and running it for 50 epochs.



Observations:

1. On comparing with phase A, we can see this model overfits really fast around the 5th epoch.
2. Yet the accuracy for validation is better than phase A which was 84% while here the accuracy reaches 92% which is a significant improvement
3. We are unable to find the possible reason for overfitting

Part C: CAPSULE NETWORKS

What is CNN?

CNN stands for Convolutional Neural Network. It is a type of Artificial Neural Network that takes into consideration a machine learning unit algorithm and perceptrons to analyze data through supervised learning. CNNs are also called as CovNets.

These CovNets consist of a sequence of layers:

- Input Layer
- Convolution Layer
- Activation Function Layer
- Pool Layer (Max Pooling / Average Pooling)
- Fully Connected Layer

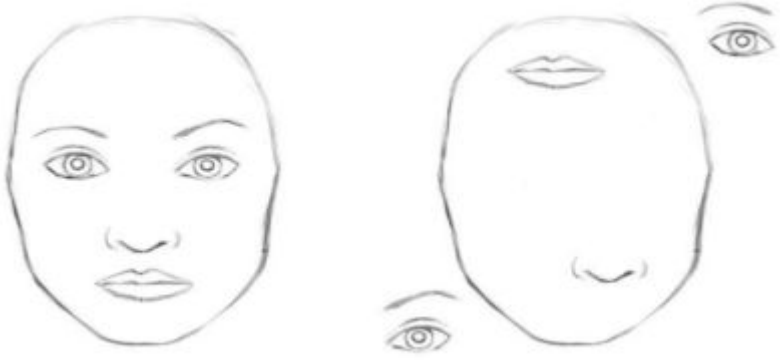
As CNNs are made up of neurons stacked together, a lot of computation would be required in calculating convolutions. Hence, the pooling processes are used to reduce the size of network layers. The initial layers in CNN first detect features like edges and then the more complex features like eyes, nose, lips (in case of an image of face) are detected. Taking all the things that the model has learnt into consideration, it makes a prediction.

Drawbacks of CNN

1. **CNNs do not decode the position and orientation of object:**

As mentioned earlier, the layers that are closer to the input detect simple features and then pass on this information to all the higher layers where all these simple features are combined into more complex features. This is done by weighted sum: activations of a preceding layer are multiplied by the following layer's weights and all these are then added and are then passed on to activation layer. CNN does not take into consideration the translational and rotational features related to an image. It looks at objects in an image as individual entities and based on the presence of all such entities, it makes the final prediction.

Eg. The presence of 2 eyes, a nose and a mouth will be predicted as a face by a CNN even though the position of these individual entities is not properly arranged.



Both the images will be predicted as face by a CNN.

2. **Lack of ability to be spatially invariant to the input data:**

Hierarchical pose relationships play a very important role in object recognition. In order to identify an object, human brain requires at the most a few dozens of images, as it understands the spatial orientation of an object in 3D. In case of CNNs, they would require a large set of data (hundreds of thousands of images) in order to achieve a good performance.

Your brain can easily recognize this is the same object, even though all photos are taken from different angles. CNNs do not have this capability

Capsule Networks:

Capsule Networks are based on inverse rendering. It tries to mimic the way a human brain learns. The problems faced in CNN is solved by the Capsule Network as it encodes the spatial information as well as the probability of an object being present.

The length of the capsule vector determines its probability of feature existence and the direction represents its pose information. CapsNets use a special type of activation function called as Squash function to normalize the magnitude of the vectors.

$$\mathbf{v}_j = \frac{||\mathbf{s}_j||^2}{1 + ||\mathbf{s}_j||^2} \frac{\mathbf{s}_j}{||\mathbf{s}_j||}$$

Squash function

Instead of using the max pooling algorithm, Capsule Networks use routing by agreement algorithm. So instead of adding a squashing function to each layer, you add it to a nested set of layers, i.e. the vector output of each capsule.

Procedure 1 Routing algorithm.

1: **procedure** ROUTING($\hat{\mathbf{u}}_{j|i}, r, l$)
2: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3: **for** r iterations **do**
4: for all capsule i in layer l: $\mathbf{c}_i \leftarrow \mathtt{softmax}(\mathbf{b}_i)$
5: for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
6: for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \mathtt{squash}(\mathbf{s}_j)$
7: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
return $\mathbf{v}_j$

algorithm of **routing-by-agreement**

How Capsules work?

1. The first step includes the matrix multiplication of input vectors with weight matrices which give us a brief idea about the spatial relationships between lower and higher-level layers.
2. The parent capsule is selected by Dynamic Routing.
3. After determining their parent capsules, summation of all the vectors is squashed between 0 and 1 while retaining their direction.

Dynamic Routing:

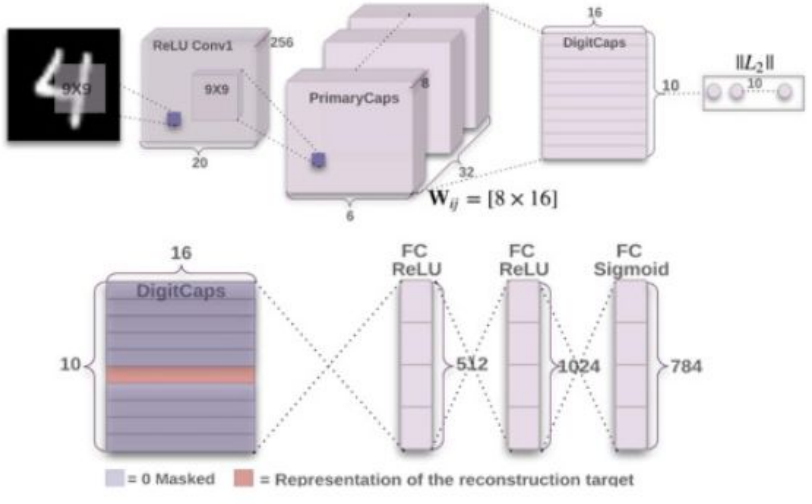
Dynamic routing is process of selecting a suitable parent capsule. In this process, the lower level capsules send data to the most suitable higher-level capsule. Selection of a suitable capsule is done by calculating the dot product between the prediction vector computed by lower layer capsule and the weighted matrix. The capsule with the highest dot product is chosen as the parent capsule.

This can be explained by the following example:

Suppose our system needs to predict whether the given image is a house or not. A CNN would be able to predict the front view of the house quite easily but will face serious difficulties while doing so if some other orientation is given as an input. It requires a huge set of training data to do so without any difficulty.

The Capsule networks on the other hand, detect the roof and the walls easily. But the presence of these two elements does not conclude that it is a house. The Capsules then analyze the common area between the 2 elements. The predictions are taken from the wall as well as the roof. These are then passed to middle level capsule and both the predictions are compared. If they match, then it is a house. This is known as Routing by agreement.

Architecture of Capsule Networks on MNIST:



This architecture is mainly made up of an Encoder and a Decoder.
Encoder: It takes the input as an image and tries to represent it as a 16-dimensional vector. It also takes into consideration the various factors necessary to render the image back.

This is done by various layers:

1.Convolution Layer: It is responsible for detecting the various features that are later analyzed by the capsules.

2.Primary (Lower) Capsule Layer: This layer contains 32 different capsules. This layer produces a 4D vector output.

3.Digit (Higher) Capsule Layer: This layer produces the 16D vector output that contains all the instantiation parameters required to build back the image.

Decoder: The decoder takes the 16D vector output from the Digit Capsule Layer and renders the image taking into consideration all the instantiation parameters.

MNIST dataset plays a crucial role in the performance of the Capsule Networks. This dataset is simple as it has only one channel unlike other datasets that are complex due to varying color, size, noise, multiple digits in single sample, etc.

Applications of CapsNets:

CapsNets are proving to solve various real-life problems in the fields of astronomy, hand-written and text-recognition, mood and emotion detection, etc. Research is still going to as there are very few proven results associated to CapsNets.

They have proved to be promising on simple datasets as compared to complex datasets. As the amount of information found in images would be more, it may throw off the capsule.