Report on Convolution Network Model Building

Kiran Kour

The objective of the Convolution Network Model:

To build a NN model with the highest accuracy when tested over the test set. Also, understanding what is the relationship between training sample size and choice of network.

Model building:

As seen in the summary table below, we built ten models with different layers, nodes, and hyper tuning parameters

Model	*Layers*	*Training Sample*	*Activation*	*Filters*	*Filter Size*	*Max-Pooling*
Model 1	4	1000	Sigmoid	32 to 256	3	Pool Size 2
Model 2	4	1000	Sigmoid	32 to 256	3	Pool Size 2
Model 3	4	1000	Sigmoid	32 to 256	3	Pool Size 2
Model 4	4	1000	Sigmoid	32 to 256	3	Pool Size 2
Model 5	4	2000	Sigmoid	32 to 256	3	Pool Size 2
Model 6	4	2000	Sigmoid	32 to 256	3	Pool Size 2
Model 7	4	1500	Sigmoid	32 to 256	3	Pool Size 2
Model 8	4	1500	Sigmoid	32 to 256	3	Pool Size 2
Model 9	3	2500	Sigmoid	32 to 256	3	Pool Size 2
Model 10	4	2500	Sigmoid	32 to 256	3	Pool Size 2
Model 11 (Pre-trained model)	1	1000	Sigmoid	32 to 256	3	Pool Size 2

Model	Optimizer	Loss	Dropout	Data Augmentation
Model 1	rmsprop	binary_crossentropy	–	–
Model 2	adam	binary_crossentropy	–	–
Model 3	adam	binary_crossentropy	0.5	Yes
Model 4	rmsprop	binary_crossentropy	0.3	Yes
Model 5	adam	binary_crossentropy	–	–
Model 6	adam	binary_crossentropy	0.5	Yes
Model 7	rmsprop	binary_crossentropy	–	–
Model 8	adam	binary_crossentropy	0.4	Yes
Model 9	adam	binary_crossentropy	–	–
Model 10	adam	binary_crossentropy	0.5	Yes
Model 11 (Pre-Trained Model)	adam(1e-5)	binary_crossentropy	0.5	Yes

Model	Epochs	Test Accuracy	Test Loss
Model 1	30	69.70%	*61.90%*
Model 2	30	70%	*59.50%*
Model 3	30	78.80%	*49.34%*
Model 4	30	50%	*69.31%*
Model 5	25	71.30%	*58.05%*
Model 6	25	79.30%	*45.02%*
Model 7	20	68.60%	*59.96%*
Model 8	20	74%	*55.14%*
Model 9	35	78.40%	*46.15%*
Model 10	35	86.50%	***31.70%***
Model 11 (Pre-Trained Model)	10	97.70%	***18.96%***

Final Observations:

- The maximum number of models was built using the **sigmoid activation function**. The sigmoid activation function is both non-linear and differentiable, which are good characteristics of the activation function. Due to the non-linearity of the sigmoid activation function, convolution networks can capture complex semantic structures and achieve high performance.

- With the **highest accuracy, model 10 and Model 11** were built using four layers, a training sample of 2500, sigmoid being the activation function. Model 10 was built using a dropout rate of 0.5, resulting in 86.50% accuracy. Similarly, building Model 11 resulted in maximum accuracy of 97.70%. Dropout is a technique where randomly selected neurons are ignored during training, thereby increasing the efficiency and accuracy of the model.

- **Model 11** was built using a **pre-trained model VGG-16**. Pre-training results are the best when training is done before the dilated con-volution model is used. When used with pre-trained weights, Keras Applications are deep learning models that can be used alongside existing deep learning models. Models like these can be used to **predict, extract, and fine-tune features**.

- The models were built using **different epochs** but increasing or decreasing epochs showed no significant increase in accuracy. The most negligible loss value and highest accuracy were achieved mainly in the initial epochs.

- Changing the number of **training samples showed considerable change** in the performance of the models. The models with training samples of 1000 and 1500 have not performed better. On the other hand when the training sample were increased the accuracy of the models increased considerably.

What is the relationship between training sample size and choice of the network?

I believe that training sample size and the choice of the network play a crucial role, but depending upon the settings in which we use them.

Training Sample Size plays a crucial role when training your model from scratch. A large enough training sample helps your model to converge better by picking up the signals. However, we might not need this in a **Pre-trained Model**.

Regarding the **Pretrained Model**, we already have the weights for that model. All we need to do is run our data through it and get the desired result. But there is a way to improve this as well. We can freeze some of its layers and train the model again with our data to perform well on our data.

How this works is like this. By freezing some layers, we are keeping the weights of those layers intact. This saves us a lot of time and computational power. If we were to train a similar model from scratch, it would take a lot of time and computational power, and there is a good chance that it might not even converge for us. So by freezing some layers helps us skip that trouble entirely.

Now we can train the last few layers of the model as this will enable the model to grasp that special signals that might only be unique to our dataset. This will ultimately give us a model that works perfectly well for us on unseen data, and we can even improve it later on with the above-mentioned method.