Description of the chosen CNN architecture

- The model follows a sequential block-style design with alternating convolution, batch normalization, ReLU activation, and max pooling layers. It progressively increases the number of feature channels while reducing spatial dimensions, culminating in a fully connected output layer.
- Key design features:
  - **Padding="same"**: Maintains spatial dimensions after convolution, ensuring consistent feature map sizes before pooling.
  - **Batch Normalization**: Stabilizes training and accelerates convergence by normalizing activations.
  - **Max Pooling**: Reduces spatial dimensions by half at each block, enabling hierarchical feature extraction and reducing computation.
  - **Channel Expansion**: Gradual increase in channels (32 → 64 → 128) allows the network to learn richer and more abstract features.
  - **Fully Connected Layer**: Maps the final flattened feature vector to 10 output classes (e.g., for CIFAR-10 or similar datasets).

Explanation of preprocessing steps:

- Random Horizontal Flip: Randomly flips the image left-to-right with a default probability of 0.5.
- Random Vertical Flip: Randomly flips the image top-to-bottom with a default probability of 0.5.
- Color Jitter: Randomly adjusts brightness and contrast (±50%) to simulate lighting variations.
- Resize: Resizes all images to a fixed size of 299x299 pixels.
- Convert to Tensor: Transforms the image from PIL format to a PyTorch tensor with shape [C, H, W].
- Normalize: Applies channel-wise normalization using mean = [0.5, 0.5, 0.5] and std = [0.5, 0.5, 0.5], which scales pixel values from [0, 1] to [-1, 1].

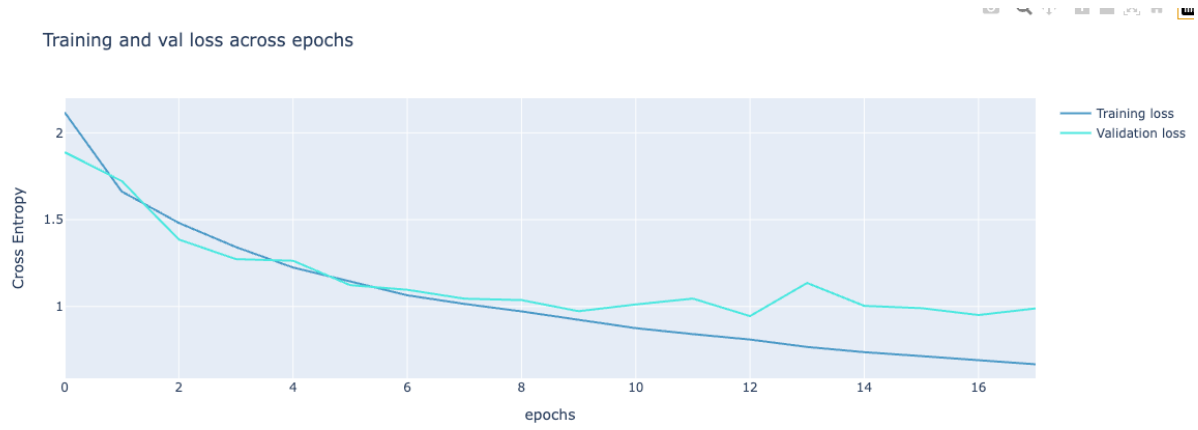
Details of training process:

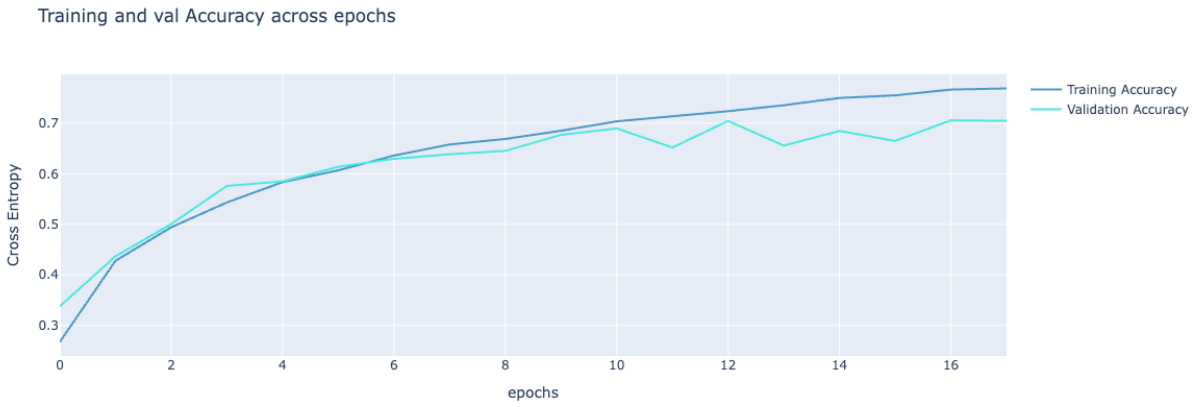
- Learning rate starts from 0.001, then is optimized by Adam optimizer.
- Batch size is 20.
- Number of epochs is initially set as 40, but early stopping was triggered after 18 epochs as the model couldn't improve the loss on the validation set for 5 consecutive times.

Results and analysis of models performance:

- CNNwithPooling:

```
Epoch [18/40], Loss: 0.6670, Acc: 0.7692, Val Loss: 0.9885, Val Acc: 0.7050, Val F1: 0.6825
    Weighted avg -- Precision: 0.7113, Recall: 0.7052, F1-score: 0.7019
    Macro avg    -- Precision: 0.6903, Recall: 0.6829, F1-score: 0.6825
```



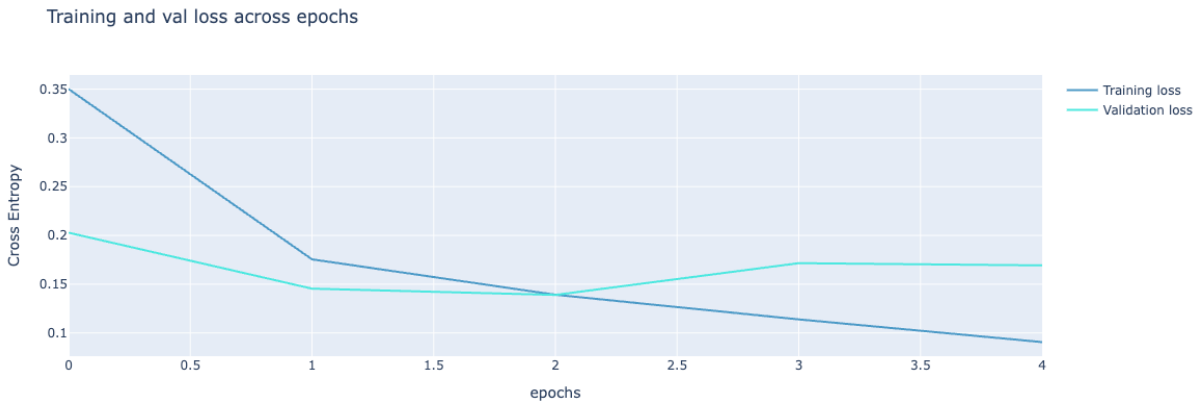


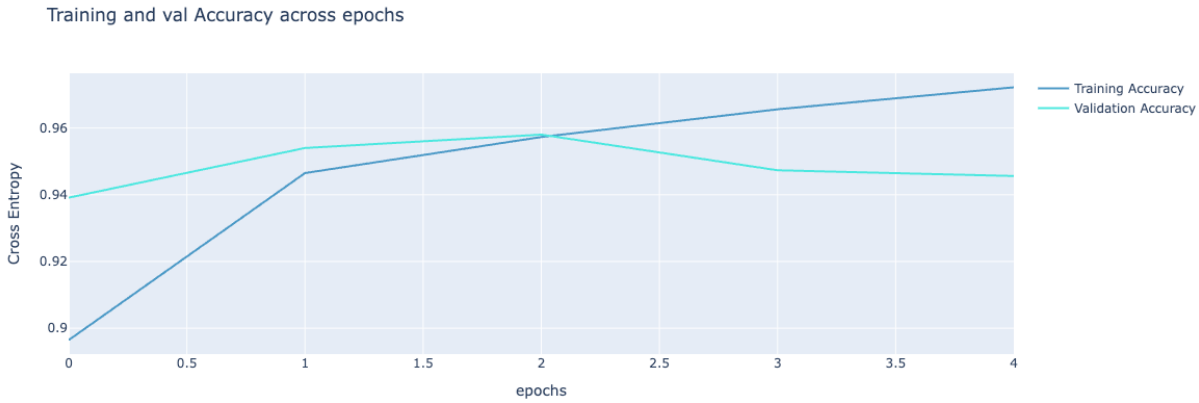
- Inception:

```
Epoch [18/40], Loss: 0.6670, Acc: 0.7692, Val Loss: 0.9885, Val Acc: 0.7050, Val F1: 0.6825
    Weighted avg -- Precision: 0.7113, Recall: 0.7052, F1-score: 0.7019
    Macro avg    -- Precision: 0.6903, Recall: 0.6829, F1-score: 0.6825
```

Confusion matrix:

```
array([[926,   4,   6,   0,   9,  13,   6,   4,   2,   3],
       [ 16, 461,  19,   0,   6,   0,  13,   7,   2,   1],
       [  1,   0, 287,   0,   0,   0,   0,   2,   0,   0],
       [  3,   0,   3, 392,   2,   1,   0,   0,  22,   0],
       [ 12,   3,   2,   0, 590,   3,   2,   3,   3,   2],
       [ 11,   0,   0,   0,   0, 321,   0,   1,   0,   1],
       [  3,   1,  10,   1,   1,   0, 342,  16,   0,   0],
       [  7,   0,   7,   0,   8,   1,   4, 334,   1,   1],
       [  3,   0,   3,   4,   3,   2,   0,   1, 947,   2],
       [  6,   0,   2,   1,   3,   4,   1,   0,   0, 356]])
```





What is your best model. Why?

- Inception model apparently outperforms CNN across all key evaluation criteria.
- Higher Accuracy & F1: Inception achieves significantly better classification performance, especially on the validation set, indicating better generalization.
- Lower Loss: Both training and validation loss are much lower, suggesting more confident and calibrated predictions.
- Balanced Class Performance: High macro F1 and precision/recall values show that Inception handles all classes well, not just the dominant ones.
- Faster Convergence: Inception reaches superior performance in just 5 epochs, while CNNWithPooling is still plateauing at epoch 18 of 40.

Insights gained from the experimentation process:

- Remember to check class imbalance in the beginning.
- Normalization is needed because:
  - It centers data around 0, which helps models converge faster.
  - It ensures consistent input distribution across batches.

- Inception model itself was trained on normalized data.