|  | **Baseline** | | **Improved version** |
| --- | --- | --- | --- |
| **Model Summary** | **Total params: 2,122,705 (8.10 MB)**  **Trainable params: 2,122,705 (8.10 MB)**  **Non-trainable params: 0 (0.00 B)** | **Total params: 1,055,089 (4.02 MB)**  **Trainable params: 1,054,977 (4.02 MB)**  **Non-trainable params: 112 (448.00 B)** | |
| **Training Graph** |  | |  |
| **Confusion Matrix** |  | |  |
|  |  | |  |
|  |  | |  |

| **Requirement** | **Baseline Template** | **Student Code (Lab6\_IPYNB)** |
| --- | --- | --- |
| **1. Layer Configuration** | 4×Conv2D (filters 8,16,32,64) → Flatten → Dense (512) → Dense (1) :contentReference[oaicite:2]{index=2}​:contentReference[oaicite:3]{index=3} | *e.g.* 5 Conv layers (16→32→64→128→256), changed kernel sizes to 5×5, added an extra Dense(256) |
| **2. Data Augmentation** | None | *e.g.* RandomFlip("horizontal"), RandomRotation(0.2), RandomZoom(0.1) |
| **3. Regularization** | None | *e.g.* BatchNormalization() after each block, Dropout(0.5) before Dense layers |
| **4. Accuracy Improvement** | N/A (baseline typically ~75% val-acc) | *e.g.* improved from 75% → 85% val-accuracy |
| **5. Evaluation Outputs** | Model summary, training/validation plots, confusion matrix, classification report :contentReference[oaicite:4]{index=4}​:contentReference[oaicite:5]{index=5} | Model summary, loss/acc curves, confusion matrix, classification report (all included) |

**What changes did you make to the CNN architecture? Why did you choose those modifications, and how did they affect model performance?**

So I make a few changes to the CNN: I put batch normalization after each conv layer and dropped in a dropout before the dense layer, and I also cut the dense layer size from 512 to 128. I did this because batch norm makes training more stable and dropout helps stop overfitting by randomly turning off neurons. As a result, the model learned faster, and its validation accuracy ended up a couple percent higher than before.

**What methods did you use to reduce overfitting (if any)? How effective were they? Explain with reference to training/validation curves.**

To deal with overfitting problem, I used data augmentation like horizontal flipping and small rotations, put in batch norm to keep the internal signals balanced, and used dropout to shake up the dense layer. When I checked the training and validation curves, I saw that the original model’s validation accuracy flattened out and even dropped, but with these tweaks the validation curve rose more steadily right along with the training curve, so it definitely worked.

**Based on your confusion matrix, what kinds of misclassifications occurred most frequently? What might be causing these errors? How would you attempt to reduce these errors?**

I noticed that in the first version the biggest oops was calling actual dogs “cats” a little more than the reverse, and in the tweaked version the most common mix-up became calling real cats “dogs” (about 856 times versus 536 dog→cat mistakes). I think this happens because some cat pictures—like fluffy fur, weird poses, or low light—look a lot like dog features to a pretty shallow CNN. To fix it I’d throw in even more varied augmentations (brightness, contrast, random crops), maybe bump up to a deeper or pre-trained backbone so it learns subtler whisker and ear shapes, and even tweak the loss or decision threshold so it pays extra attention to those tricky cat shots.