**FASHION\_MNIST**

* **MLP:** Starting performance yielded a test accuracy of 0.8899000287055969.

|  |  |
| --- | --- |
| **Parameter name** | **Value** |
| Layers (exluding output) | 2 |
| Units | 512 |
| Dropout | 0.2 each |
| Activation | SiLu, SiLu, Softmax |
| Optimiser | RMSPROP |
| Epochs | 20 |

* Increased dropout for regularisation
* Increased depth to 4 layers
* Added batch normalisation
* Changed optimiser to ADAM

This improved the accuracy to 0.9010000228881836

* **CNN:** Starting performance yielded a test accuracy of 0.72589999437332150.7

|  |  |
| --- | --- |
| **Parameter name** | **Value** |
| Layers (exluding output) | 4 |
| Units | 32,64,128 |
| Dropout | 0.25,0.5 |
| Activation | ReLu |
| Optimiser | Adedelta |
| Epochs | 20 |

* Adding more Conv2D layers (3 per block) and changing optimiser to Adam elevates accuracy all the way to 0.9373000264167786

**CIFAR-10**

* **MLP:** Proposed MLP architecture achieved a test accuracy of 0.5302000045776367
* **CNN:** Proposed CNN architecture achieved a test accuracy of 0.8593999743461609

This goes on to show that MLP is severely limited with regard to image classification. Therefore, our CNN implementation is able to generalise much better (even though on CIFAR-10 it still falls significantly short of its Fashion MNIST performance)

**TELL THE TIME NETWORK**

* The first approach we tried is arguably the most intuitive one with 720-way classification.

1. Data processing

First, we transformed the original (hour,minute) format labels into combined hour\*60 + minute label space. Afterwards we split the dataset 0.8/0.1/0.1 (just like before) and prefetched the data, which – as per official TensorFlow guide – reduces training time

*Prefetching overlaps the preprocessing and model execution of a training step. While the model is executing training step s, the input pipeline is reading the data for step s+1. Doing so reduces the step time to the maximum (as opposed to the sum) of the training and the time it takes to extract the data. (*[*https://www.tensorflow.org/guide/data\_performance*](https://www.tensorflow.org/guide/data_performance)*)*

1. *Network Architecture*

As for the neural network architecture, we decided to make it sufficiently deep in order for its capacity to be suitable for coping with this many different classes. This resulted in 4 separate blocks being formed:

* Block 1 for low level feature extraction:

2x convolution layer of size 64 with (3,3) kernel and ReLu activation

Followed by (2,2) max pooling and a 0.2 dropout layer

* Block 2 for mid-level features:

The same but with convolutions of size 128

* Block 3 for high-level features:

One convolutional layer of size 256 with (3,3) kernel

Followed by (2,2) max pooling and a 0.2 dropout

* Block 4 for dense layers

Input flattening followed by a 1024-size dense layer and 512-size dense layer, with 0.4 dropout after each

* Output layer with softmax activation appropriate for multi-class classfication

1. Training and compilation

The loss function of choice is sparse categorical crossentropy as our true labels are provided in the form of integers (as opposed to one-hot encoding).

Further, we used 3 callback functions to better steer the learning process:

* **ReduceLROnPlateau** - Adaptive Learning Rate: we want to make good progress at the beginning (fast learning), but as we approach the theoretical optimum slower LR becomes more desirable for escaping plateaux. If validation loss does not improve for 3 consecutive epochs, we multiply it by a factor of 0.5
* **EarlyStopping** - Prevents Overfitting: In case of 7 consecutive epochs without improvement on validation loss, we stop training to avoid overfitting
* **ModelCheckpoint** - Saves Best Model: As soon as a validation loss improvement is registered, the incumbent models get saved as a back-up against potential crashes.

We train our model over 30 epochs

1. Evaluation

We evaluate our model on the test set by setting its predictions (converted back actual hours/minute values) against the true labels.

As for the common-sense metric, we compute it via *calculate\_time\_difference\_minutes(true\_hours, true\_minutes, pred\_hours, pred\_minutes)*

function that handles clock wrap-around by taking a minimum of error and 720-error (12 hours \* 60 minutes).

The evaluation results are as follows:

Detailed Accuracy:

Hour accuracy: 0.6756 (67.56%)

Minute accuracy: 0.2289 (22.89%)

Exact match (both correct): 0.2200 (22.00%)

"Common Sense" Time Difference Accuracy:

Mean absolute error: 48.05 minutes

Median absolute error: 1.00 minutes

Std deviation: 87.62 minutes

Worst case (max error): 359 minutes

(plots etc.)

* Our second approach was to frame it as a multi-head output problem, i.e. to treat hours and minutes separately.

1. Data processing:

We treat labels as tuples:

dataset = tf.data.Dataset.from\_tensor\_slices((data, (hour\_labels, minute\_labels)))

Data split and prefetching as above

1. Network architecture:

CNN backbone the same as with multi-class classification, the difference manifests itself in the dual nature of dense layers:

**Hour head**: Specialized branch with 128 units → 12 classes

**Minute head**: Specialized branch with 256 units → 60 classes (larger because it's a harder task)

Output is naturally also twofold: [hour\_output, minute\_output]

1. Training and compilation:

Both output heads use sparse categorical crossentropy and the same callback functions as the model above.

1. Evaluation:

Method same as above, yielding the following results:

Exact match (both correct): 0.3983 (39.83%)

"Common Sense" Time Difference Accuracy:

Mean absolute error: 4.82 minutes

Median absolute error: 1.00 minutes

Std deviation: 16.04 minutes

Worst case (max error): 237 minutes

* Final approach: regression with angle conversion

The most sophisticated way of addressing this problem is to convert all the hour and minute labels into angles and then convert those angles into sin/cos function (to handle circularity issues).

1. Data processing: