# MultiLayer Perceptron

There are four classes of ECG; **HB** is patients with abnormal heart beats, **Normal** is a normal persons ECG, **PMI** is for patients that have a history of Myocadial Infraction (**MI**), and **MI** is for patients suffering from MI.

**Configuration:**

1. **Input Layer:**
   * keras.layers.Flatten(input\_shape=[180, 230]): This layer flattens the input images, which are originally 180x230 pixels, into a 1D array. This is necessary for feeding the data into the subsequent dense layers.
2. **Hidden Layers:**
   * The first hidden layer has 200 neurons and uses the ReLU (Rectified Linear Unit) activation function.
   * Batch normalization is applied after the first hidden layer to help stabilize and speed up training by normalizing the activations.
   * Dropout is used to prevent overfitting by randomly dropping out 70% of the neurons during training. This forces the network to learn more robust features.
   * The pattern of Dense layer, Batch Normalization, and Dropout is repeated with different neuron counts and dropout rates for the second and third hidden layers. The second layer has 100 neurons with a dropout rate of 0.6, and the third layer has 32 neurons with a dropout rate of 0.6.
3. **Output Layer:**
   * The output layer has 4 neurons, corresponding to the 4 classes in the dataset ('HB', 'MI', 'Normal', 'PMI'). The softmax activation function ensures that the output is a probability distribution over the classes.
4. **Compilation:**
   * model.compile(loss="sparse\_categorical\_crossentropy", optimizer="sgd", metrics=["accuracy"]):
     + loss="sparse\_categorical\_crossentropy" is the appropriate loss function for multi-class classification with integer labels.
     + optimizer="sgd" uses the stochastic gradient descent optimizer for training the model.
     + metrics=["accuracy"] specifies that accuracy will be used to evaluate the model's performance.
5. **Training:**
   * history = model.fit(X\_train, y\_train, epochs=40, validation\_data=(X\_valid, y\_valid), callbacks=[early\_stopping]):
     + The model is trained for a maximum of 40 epochs using the training data (X\_train, y\_train).
     + The validation data (X\_valid, y\_valid) is used to monitor the model's performance during training and prevent overfitting.
     + Early stopping is implemented to stop training if the validation loss does not improve for 5 consecutive epochs.

A screenshot of a computer program

Description automatically generated

**Results:**

A blue squares with white text

Description automatically generated

**Overall:**

* The model seems to be quite accurate, especially at identifying MI and Normal ECGs.
* Confusion between HB and other categories, indicating potential challenges in accurately classifying patients with abnormal heartbeats.

**Specific Observations:**

* **MI:** The model is excellent at identifying MI, with 609 correct predictions and no misclassifications. This is crucial for prompt and accurate diagnosis of this serious condition.
* **Normal:** Similarly, the model performs very well with Normal ECGs, correctly classifying 529 cases. There's some confusion with HB (56 cases) and PMI (47 cases)
* **PMI:** The model correctly identifies 306 PMI cases. A significant number (75) are misclassified as HB. This suggests the model might struggle to differentiate between past and current heart issues, particularly when it comes to abnormal heartbeats.
* **HB:** This is where the model seems to have the most difficulty. While 370 HB cases are correctly classified, there's significant confusion with all other categories:
  + 94 misclassified as MI
  + 78 misclassified as Normal
  + 12 misclassified as PMI

**1. HB (Abnormal Heartbeats)**

* **Precision (0.78):** When the model predicts HB, it's correct 78% of the time. This means there's a 22% chance it incorrectly labels another condition as HB (False Positive).
* **Recall (0.67):** The model correctly identifies 67% of actual HB cases (True Positives). This implies it misses 33% of HB cases (False Negatives), potentially leading to missed diagnoses.
* **F1-score (0.72):** A balance between precision and recall, suggesting moderate overall performance in identifying HB.

The model is okay at identifying abnormal heartbeats, but it has a tendency to over-diagnose them (false positives) and also misses a significant portion of actual cases (false negatives).

A screenshot of a computer

Description automatically generated

**2. MI (Myocardial Infarction)**

* **Precision (0.97):** When the model predicts MI, it's almost always correct (97% of the time). There's a very low chance of false positives.
* **Recall (1.00):** The model captures every single instance of MI in the dataset (True Positives), meaning no cases are missed (no False Negatives).
* **F1-score (0.99):** Near-perfect score, reflecting excellent performance in MI detection.

The model is extremely reliable at identifying MI, with very high accuracy and no missed cases. This is crucial for this critical condition.

**3. Normal**

* **Precision (0.76):** When the model predicts a Normal ECG, it's correct 76% of the time. There's a 24% chance of a false positive, meaning it might incorrectly label an abnormal ECG as normal.
* **Recall (0.84):** The model correctly identifies 84% of actual Normal ECGs (True Positives). It misses 16% of normal cases (False Negatives).
* **F1-score (0.79):** Good overall performance in identifying Normal ECGs.

The model is good at identifying normal ECGs, but it can sometimes misclassify abnormal ones as normal, which could be a concern.

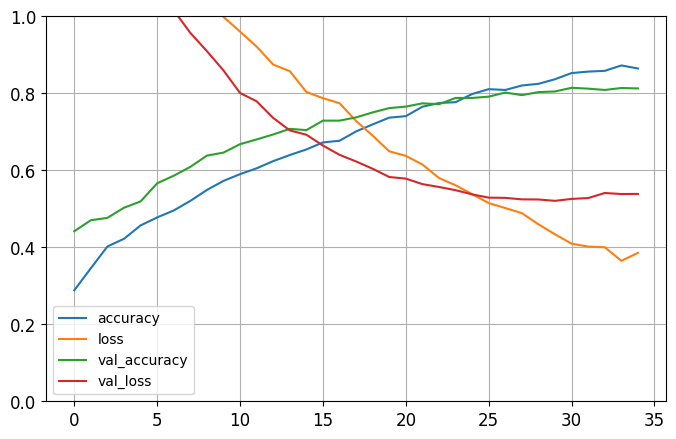
**4. PMI (Post Myocardial Infarction)**

* **Precision (0.71):** When the model predicts PMI, it's correct 71% of the time. There's a 29% chance of a false positive.
* **Recall (0.71):** The model correctly identifies 71% of actual PMI cases (True Positives). It misses 29% of PMI cases (False Negatives).
* **F1-score (0.71):** Moderate performance in identifying PMI, with room for improvement.

The model has a moderate ability to identify patients with a history of MI. It tends to both misclassify other conditions as PMI and miss actual PMI cases

**Discussion:**

The model had an issue with overfitting, as can be seen with the graph accuracy and loss graph below.



The model starts to learn the images too well. Which starts to decrease the accuracy of the model on unseen data and increases the loss. That is why dropout was introduces along with early stopping to prevent the validation loss from increasing. Batch normalization was also incorporated to improve the training.

**Dropout**

* Randomly "drops out" (sets to zero) a fraction of the neurons in a layer during each training step. This means that different subsets of neurons are active in each step.
  + **Prevents overfitting:** By forcing the network to learn with different combinations of neurons, it prevents the network from relying too heavily on any single neuron or small group of neurons. This makes the model more robust and generalizes better to unseen data.
  + **Ensembles multiple networks:** Dropout can be seen as training an ensemble of multiple smaller networks, as each dropout configuration effectively creates a different subnetwork. This ensemble effect can improve overall performance.

**Batch Normalization**

Normalizes the activations (outputs) of a layer for each mini-batch during training. It essentially standardizes the inputs to each layer, ensuring they have zero mean and unit variance.

* + **Faster training:** By stabilizing the inputs to each layer, batch normalization allows for the use of higher learning rates and speeds up the training process.
  + **Improved gradient flow:** It helps mitigate the vanishing gradient problem, making it easier to train deeper networks.

**Discussion**

* Improvement to processing images could have improved the model. Some of the images had solid lines for the signals while others had empty space between the top and the bottom of the image. Creating variance in the signals, making it more difficult for the model to learn. Some images had lines that were not fully processed.
* The overfitting problem should be further looked at to improve the training of the model